MASTER EN SCIENCES INFORMATIQUES

Intégration de spécifications de bases de données

Mynsberghe, Vincent; Vanackere, Patrick

Award date:
1991

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES N.D. DE LA PAIX NAMUR

INSTITUT D'INFORMATIQUE

INTEGRATION DE SPECIFICATIONS DE BASES DE DONNEES

Vincent MYNSBERGHE
Patrick VANACKERE

Mémoire présenté sous la direction du
Professeur Jean-Luc HAINAUT
Pour l'obtention du titre de
Licencié et Maître en Informatique

Année académique 1990-1991

Abstract

The integration of database schemata is seen as a problem which is at the same time crucial and difficult to solve. Crucial problem because it corresponds to an important conceptual modelisation process when more than one designer contribute to the single global shema. Difficult problem because it consists of comparisons of two specifications sets (i.e. the determination of common, specific and conflicting parts) which representation (selected object: entity, relationship, attribute; denomination) can be conflicting, then of transformations which allow to merge, without redundancy, the two sets. This problem has already been widely studied in the literature, but no general solution seems to emerge yet.

The aim of this thesis consists of the synthesis of the problem then of the definition of a coherent subset of this problem. We will propose a realistic model and methodology that we will validate by practical studies. We will implement this solution in the database design CASE tools TRAMIS.

Résumé

L'intégration de schémas de bases de données est perçue comme un problème à la fois crucial et difficile à résoudre. Problème crucial car il correspond à un processus important de la modélisation conceptuelle lorsque plus d'un spécificateur apporte sa contribution au schéma global unique. Problème difficile car il consiste à comparer deux ensembles de spécifications (càd déterminer ce qui est commun, ce qui est spécifique, ce qui est contradictoire) dont les représentations (objet choisi: entité, association, attribut; dénomination) peuvent être divergentes, puis à opérer des transformations qui permettent de fusionner, sans redondance, les deux ensembles. Ce problème a déjà été largement étudié dans la littérature, sans qu'une solution générale émerge réellement.

Le but du mémoire consiste à réaliser une synthèse de la question (ou plutôt à actualiser celle qui est réalisée dans un mémoire précédent), puis à définir un sous-ensemble cohérent du problème. Nous proposerons un modèle et une démarche réalistes que nous validerons par des études pratiques. Nous implanterons cette solution dans l'atelier de conception de bases de données TRAMIS.

Remerciements

Nous tenons à exprimer nos plus sincères remerciements à l'égard de toutes les personnes qui ont, de près ou de loin, contribué à l'aboutissement de ce travail.

Nous remercions plus particulièrement:

Monsieur Jean-Luc Hainaut, promoteur de ce mémoire, qui nous a guidé et conseillé au cours de cette année académique.

Monsieur Pierre Tornier et toute l'équipe CCIC, pour leur accueil chaleureux et pour l'expérience qu'ils nous ont permis d'acquérir tout au long du stage que nous avons effectué dans le centre européen de DIGITAL à Ferney-Voltaire.

Messieurs Bernard Decuyper et Olivier Marchand, membres de l'équipe TRAMIS, pour leurs précieux conseils lors de la réalisation de l'application d'intégration dans l'atelier logiciel TRAMIS.

Toutes les personnes de notre entourage qui nous ont encouragé et soutenu durant nos études universitaires.

Vincent Mynsberghe
Patrick Vanackere

TABLE DES MATIÈRES

Abstract i

Résumé iii

Remerciements v

Chapitre 1 INTRODUCTION 1–1

1.1 Position du problème de l'intégration 1–1

1.2 Objectifs et Organisation du mémoire 1–2
1.2.1 Objectifs du mémoire 1–2
1.2.2 Organisation du mémoire 1–2

1.3 Historique du problème 1–4

1.4 Les deux approches 1–5
1.4.1 Intégration de vues 1–5
1.4.2 Intégration de bases de données existantes 1–5

Chapitre 2 ETAT DE L'ART SUR L'INTEGRATION DES VUES 2–1

2.1 Introduction 2–1

2.2 La démarche de *[Elmasri & Wiederhold 79]* 2–2
2.2.1 Le modèle de données 2–2
2.2.2 La démarche 2–5

2.3 La démarche de *[Yao & Al 82]* 2–7
2.3.1 Le modèle de données 2–7
2.3.2 La démarche 2–7

2.4 La démarche de *[Batini & Lenzerini 84]* 2–9
2.4.1 Le modèle de données 2–9
2.4.2 La démarche 2–11

2.5 La démarche de *[Dayal & Hwang 84]* 2–16
2.5.1 Le modèle de données 2–16
2.5.2 La démarche 2–17

2.6 La démarche de *[Biskup & Convent 86]* 2–20
2.6.1 Le modèle de données 2–21
2.6.2 La démarche 2–21

2.7 La démarche de *[Motro 87]* 2–22
2.7.1 Le modèle de données 2–23
2.7.2 La démarche 2–24

2.8 La démarche de *[Larson & Al 89]* 2–27
2.8.1 Le modèle de données 2–27
2.8.2 La démarche 2–28

2.9 La démarche de *[Bouzeghoub & Al 90]* 2–31
2.9.1 Le modèle de données 2–32
2.9.2 La démarche 2–33

2.10 La démarche de *[Spaccapietra & Parent 90]* 2–35
2.10.1 Le modèle de données 2–37
2.10.2 La démarche 2–37

Chapitre 3 SYNTHESE DES PROBLEMES SUR L'INTEGRATION DES VUES 3–1

3.1 Introduction 3–1

3.2 Les modèles conceptuels utilisés 3–1
3.2.1 Le modèle Entite-Association 3–1
3.2.2 Le modèle Relationnel 3–2
3.2.3 Le modèle Fonctionnel 3–4

3.3 Les stratégies d'intégration 3–6
3.3.1 La stratégie binaire 3–6
3.3.2 La stratégie n-aire 3–7

3.4 Les écoles d'intégration 3–8
3.4.1 Introduction 3–8
3.4.2 Intégration dynamique 3–9
3.4.3 Intégration virtuelle partielle 3–10
3.4.4 Intégration virtuelle globale 3–12
3.4.5 Intégration physique globale 3–13

3.5 Démarche générale du processus d'intégration 3–15
3.5.1 Modélisation 3–16
3.5.2 Préintégration 3–16
3.5.3 Mapping structurel 3–16
3.5.4 Intégration 3–17
3.5.5 Restructuration 3–18
3.5.6 Mapping opérationnel 3–18

3.6 Les types de démarches 3–18
3.6.1 Démarche manuelle 3–18
3.6.2 Démarche interactive 3–19
3.6.3 Démarche automatique 3–19

3.7 Les problèmes de l'intégration 3–19
3.7.1 Ordre d'intégration 3–20
3.7.2 Problèmes de terminologie 3–20
3.7.3 Conflit de structures 3–22
3.7.4 Recouvrement de classes 3–22
3.7.5 Conflit entre spécifications 3–23
3.7.6 Restructuration et Enrichissement 3–24
3.7.7 Taille des schémas 3–24

Chapitre 4 COMPARAISON ET CRITIQUE DES DEMARCHES 4–1

4.1 Introduction 4–1

4.2 Les écoles d'intégration 4–1

4.3 Le modèle de données 4–2

4.4 Position de l'intégration par rapport à la conception 4–3

4.5 Objets en entrée et en sortie 4–3

4.6 Etapes de l'intégration couvertes 4–4

4.7 Stratégie d'intégration 4–5

4.8 Détection de conflits 4–6

4.9 Technique de fusion choisie 4–7

4.10 Opérations après fusion 4–8

4.11 Nature du processus d'intégration 4–8

4.12 Bilan et Conclusions des démarches actuelles 4–9

Chapitre 5 METHODOLOGIE GENERALE D'INTEGRATION POUR UN ATELIER LOGICIEL 5–1

5.1 Introduction 5–1

5.2 Définition et Objectifs d'un atelier logiciel 5–1

5.3 Environnement de travail 5–2

5.4 Méthodologie générale d'intégration 5–4
5.4.1 Stratégie d'intégration pour un atelier logiciel 5–4
5.4.1.1 Applicabilité 5–4
5.4.1.2 Intégration binaire 5–4
5.4.1.3 Politique d'intégration 5–6
5.4.1.4 Détection des homonymes 5–6
5.4.2 La typologie des correspondances 5–7
5.4.3 Le Mapping Structurel 5–11
5.4.4 Cohérence du mapping structurel 5–14

5.5 Les fonctions d'intégration nécessaires 5–15
5.5.1 Transformations d'un objet en une entité 5–17
5.5.2 Intégration d'une entité 5–19
5.5.3 Intégration d'une association 5–23
5.5.4 Intégration d'un attribut 5–30

5.6 Algorithme général d'intégration 5–35
5.6.1 L'algorithme 5–35
5.6.2 La fonction "INTEGRER" 5–35

5.7 Quelques exemples 5–39

5.8 Noyau minimal des fonctions d'intégration pour un atelier logiciel 5–46

Chapitre 6 IMPLEMENTATION DES FONCTIONS NECESSAIRES A UN ATELIER LOGICIEL: TRAMIS 6–1

6.1 Introduction 6–1

6.2 L'atelier logiciel existant 6–1
6.2.1 Présentation de l'environnement *MS-WINDOWS* 6–1
6.2.1.1 Contenu de MS-WINDOWS 6–1
6.2.1.2 La place de WINDOWS 6–3
6.2.1.3 Qu'apporte WINDOWS ? 6–4
6.2.2 Présentation de l'environnement *TRAMIS* 6–5

6.3 Conception du module d'intégration 6–7
6.3.1 Le méta-schéma de l'atelier TRAMIS/MASTER 6–7
6.3.2 Dynamique générale de l'ensemble des modules 6–9
6.3.3 Description des traitements de l'algorithme d'intégration 6–11
6.3.4 Méta-algorithme ADL des fonctions d'intégration 6–14
6.3.5 Macros fonctions employées 6–18
6.3.6 Limitations de l'algorithme 6–19

6.4 Conclusion 6–19

Chapitre 7 CONCLUSION GENERALE 7–1

Annexe A MÉTA-SCHÉMA DE L'ATELIER TRAMIS/MASTER A–1

Annexe B MÉTA-ALGORITHME DU MODULE D'INTÉGRATION B–1

B.1 Fonction Principale du module d'intégration B–1

B.2 Fonctions EE B–3

B.3 Fonctions ER B–5

B.4 Fonctions EA B–8

B.5 Fonctions EN B–11

B.6 Fonctions RR B–12

B.7 Fonctions RE B–15

B.8 Fonctions RA B–18

B.9 Fonctions RN B–22

B.10 Fonctions AA B–24

B.11 Fonctions AE B–27

B.12 Fonctions AR B–30

B.13 Fonctions AN B–33

B.14 Fonctions complémentaires B–35

Annexe C CODE SOURCE DU MODULE D'INTÉGRATION C–1

Annexe D SESSION D'INTÉGRATION DANS TRAMIS D–1

D.1 Exemple de session d'intégration dans TRAMIS D–1

BIBLIOGRAPHIE Glossaire–1

EXEMPLES

2–1 L'opérateur REMOVE 2–9
2–2 L'opérateur MERGE 2–9
3–1 Réparation de voitures accidentées 3–2
3–2 Le modèle d'intégration virtuelle de *[Motro 87]* 3–11
3–3 Ambiguïté du texte 3–23
5–1 Illustration d'un mapping structurel 5–14

FIGURES

2–1 Types de connexions 2–4
2–2 Les quatre représentations possibles du réel 2–6
2–3 Exemple de schéma Entité/Association 2–10
2–4 Exemple d'anomalie de noms 2–12
2–5 Restructuration du schéma intégré 2–15
2–6 Les principaux concepts du modèle FDM et leur représentation graphique 2–17
2–7 Exemple de schéma fonctionnel 2–17
2–8 Exemple d'intégration par l'emploi du mécanisme de généralisation 2–19
2–9 Exemple du modèle fonctionnel de *[Motro 87]* 2–23
2–10 Les opérateurs MEET et JOIN 2–25
2–11 Les opérateurs FOLD, TELESCOPE et AGGREGATE 2–26
2–12 Exemple de schéma ECR 2–28
2–13 Integration de types d'associations de degrés différents 2–30
2–14 Représentation en MORSE des concepts du modèle Entité/Association 2–32
2–15 Représentation graphique d'un réseau sémantique 2–33
2–16 Algorithme d'intégration des objets des vues 2–36
3–1 Schéma entité/association 3–3
3–2 Schéma relationnel 3–4
3–3 Schéma fonctionnel 3–5
3–4 La stratégie binaire 3–6
3–5 La stratégie n-aire 3–7
3–6 Schéma de l'intégration dynamique 3–9
3–7 Schéma de l'intégration virtuelle partielle 3–11
3–8 Schéma de l'intégration virtuelle globale 3–13
3–9 Schéma de l'intégration physique globale 3–14
3–10 Démarche générale du processus d'intégration 3–15
3–11 Importance de l'ordre d'intégration 3–21

3–12 Les différents recouvrements possibles 3–23
5–1 Phases de conception de la base de données 5–5
5–2 Les schémas Commande-Client 5–14
5–3 Incohérence au niveau du mapping structurel 5–15
5–4 Transformation d'une association en une entité 5–18
5–5 Transformation d'un attribut d'une entité en une entité 5–19
5–6 Sous-schéma 1 5–43
5–7 Sous-schéma 2 5–43
5–8 Sous-schéma 3 5–44
5–9 Le schéma intermédiaire 5–45
5–10 Résultat final des deux phases d'intégration 5–47
5–11 Noyau minimal 5–48
6–1 Les ensembles fonctionnels de WINDOWS 6–2
6–2 Place de WINDOWS dans le PC 6–4
6–3 L'architecture de TRAMIS 6–6
6–4 Les concepts utilisés 6–10
6–5 Dynamique de l'algorithme 6–11
6–6 Détail des fonctions avec correspondances 6–12
6–7 Détail des fonctions sans correspondance 6–13
D–1 Début du processus d'intégration D–1
D–2 Choix du schéma primaire D–2
D–3 Demande du correspondant D–3
D–4 Demande du nom D–3
D–5 Demande de permutation des nom D–4
D–6 Résultat après la première correspondance D–4
D–7 Comment demander à voir les correspondance D–5
D–8 La correspondance introduite D–6
D–9 Résultat final D–7
D–10 La liste des correspondances introduites D–8

TABLEAUX

2–1 Les caractéristiques des concepts de base du modèle E/A 2–11
4–1 Ecole d'intégration 4–2
4–2 Modèles de données utilisés 4–2
4–3 Position du processus d'intégration 4–3
4–4 Entrées/Sorties 4–4
4–5 Etapes d'intégration 4–5
4–6 Stratégie d'intégration 4–6
4–7 Détection des conflits 4–6
4–8 Fusion 4–7
4–9 Restructuration du schéma intégré 4–8
4–10 Nature du processus d'intégration 4–9

CHAPITRE 1

INTRODUCTION

1.1 Position du problème de l'intégration

La conception d'un schéma de base de données est généralement découpée en trois phases d'après la norme définie par *[ANSI 75]* et maintenant admise par tous.
"Dans une première phase, dite **phase conceptuelle**, on procède au rassemblement de schémas locaux qui proviennent de l'analyse des besoins de chaque secteur concerné par la future base de données. On produit ainsi un seul schéma conceptuel de l'application à réaliser qu'on complète par des spécifications.
La deuxième phase, appelée **phase logique**, consiste à traduire le schéma conceptuel en un schéma logique: on transforme les concepts spécifiés au niveau conceptuel dans le langage supporté par le système de gestion de bases de données (SGBD) qu'on va utiliser.
La troisième phase, intitulée **phase physique** ou phase de conception physique, permet la définition du stockage physique des données et des méthodes d'accès à ces données. Les personnes chargées de cette phase y indiquent aussi les paramètres physiques des données (lors de la définition du format des enregistrements) et l'usage des données (qui dépend du mécanisme d'accès qui leur est associé: si une donnée doit servir pour retrouver un objet, il faut définir celle-ci comme une clé d'accès (qui peut faire partie d'un index, . . .) pour que le SGBD puisse répondre au souhait de l'utilisateur)".

Dans ce mémoire, nous nous intéresserons uniquement à la première phase de la conception.

Cette phase est fondamentale, elle aboutit à la définition du schéma conceptuel global de l'application. "Ce schéma doit refléter la structure des objets jouant un rôle dans l'application et les liens entre les différents objets. Il doit être approuvé de toutes les parties prenantes. Pour obtenir ce consensus, deux méthodes sont possibles:

— soit le concepteur étudie l'ensemble de l'application et élabore un schéma conceptuel qu'il modifie jusqu'à satisfaction de toutes les personnes concernées,

— soit, pour chaque personne ou groupe de personnes concerné, il construit un schéma[1] décrivant la partie de l'application pour cette personne. Puis il réalise la fusion de l'ensemble des schémas ainsi obtenus." *[Comyn-Wattiau 90]*

Chaque approche a son intérêt. La première fait en sorte que ce soit la même personne qui traite le problème et qu'elle ait une connaissance globale du sujet, ce qui évite un tas de problèmes de communication.

[1] Il se peut dans certains cas que ce soient plusieurs personnes qui travaillent en parallèle et produisent chacune une partie du schéma final.

Cependant, pour avoir une connaissance globale, cela implique que la taille de l'application soit "raisonnable". Dans le cas de grosses applications, la deuxième approche s'impose d'elle-même. Dans cette approche, le travail en parallèle n'apporte pas un gain de temps si évident qu'il pourrait paraître au premier abord: si les schémas sont réalisés plus vite, il faut les mettre en commun par la suite. Or les gens peuvent avoir eu des perceptions différentes et il faut détecter les ambiguïtés, similitudes et inexactitudes contenues dans ces schémas, ce qui nécessite une bonne communication entre les différents concepteurs.

Le processus d'intégration des vues consiste, à partir d'un certain nombre de schémas modélisant des parties d'une application (ou parfois de schémas d'applications qu'on a envie de centraliser), de les regrouper en un schéma conceptuel global. Cela implique de détecter les parties identiques pour éviter des redondances dans les données à enregistrer. Il faut aussi repérer les parties en conflits pour éviter d'introduire des incohérences dans la base de données qui tôt ou tard pourront produire des résultats erronés[2].

Il y a quelques années, ce processus se faisait encore manuellement, ce qui pouvait être source d'erreurs (incompréhension entre concepteurs, oubli d'un aspect dans un schéma trop vaste, . . .). C'est pourquoi bon nombre d'auteurs ont cherché une solution automatisable à ce problème.

1.2 Objectifs et Organisation du mémoire

Nous présentons maintenant la raison de ce mémoire et comment nous avons décidé de le découper.

1.2.1 Objectifs du mémoire

Comme indiqué dans la section 1.1, le problème de l'intégration de schémas conceptuels de bases de données est très difficile. Il existe à l'heure actuelle peu d'outils pour le résoudre.

Notre mémoire a pour objectif de réaliser une étude comparative de différentes recherches accomplies dans le domaine de l'intégration et de réactualiser l'enquête de référence réalisée en cette matière par *[Batini & Al 86]*. Il doit aussi servir à prolonger les travaux accomplis dans un précédent mémoire (*[Celis & Deudon 88]*). Sur base de ces acquis, nous élaborerons une méthodologie pour la résolution du problème de l'intégration dans un environnement d'atelier logiciel.

A partir de cette solution déduite, nous proposerons une solution limitée de résolution du problème de l'intégration qui sera incorporée dans un atelier logiciel et qui permettra d'intégrer plusieurs schémas en un schéma conceptuel global.

1.2.2 Organisation du mémoire

Le présent chapitre (INTRODUCTION) présente quelque peu le problème puis décrit les objectifs du mémoire. Il se poursuit par un survol historique de l'évolution du problème et des recherches accomplies en cette matière et se termine par la présentation des deux approches possibles pour l'intégration de schémas.

[2] Ces erreurs peuvent être néfastes aux utilisateurs qui emploient cette base de données principalement dans l'optique de la prise de décisions stratégiques.

Le deuxième chapitre (ETAT DE L'ART SUR L'INTEGRATION DES VUES) présentera quelques études réalisées dans le domaine de l'intégration de schémas de bases de données. Nous y découvrirons différentes approches qui prouveront à suffisance que ce problème est loin d'être trivial, ce qui est une des raisons pour lesquelles il y a actuellement peu de solutions et encore moins d'outils d'aide à l'intégration.

Le troisième chapitre (SYNTHESE DES PROBLEMES SUR L'INTEGRATION DES VUES) fera une synthèse des aspects abordés lors du Chapitre 2 et donnera une brève descriptions de chaque concept, à savoir la méthode employée, les stratégies possibles, les diverses écoles[3] d'intégrations, le processus d'intégration en lui-même et les différentes façons de l'appliquer. Nous terminerons par les différents problèmes posés par l'intégration et étudiés (plus ou moins en profondeur) par les divers auteurs présentés lors du Chapitre 2.

Le quatrième chapitre (COMPARAISON ET CRITIQUE DES DEMARCHES) proposera une comparaison des différentes méthodes vues lors du Chapitre 2 sur les aspects suivants:

- l'école à laquelle elles appartiennent,
- le modèle de description de schémas choisi,
- la place où elles se situent dans le processus d'intégration,
- les données nécessaires en amont et en aval du processus,
- la stratégie d'intégration adoptée,
- les conflits qu'elles étudient et le moyen de les détecter,
- les différentes étapes du processus d'intégration de chacune,
- la technique de fusion de schémas choisie,
- les éventuelles opérations de restructuration nécessaires après le processus d'intégration, et finalement
- le type de démarche employé pour intégrer des schémas.

Cette comparaison appellera une petite synthèse des similitudes et différences découvertes.

Dans le cinquième chapitre (METHODOLOGIE GENERALE D'INTEGRATION POUR UN ATELIER LOGICIEL), nous définirons et justifierons notre méthodologie générale dans le cadre d'un atelier logiciel. Nous décrirons le contexte de notre méthodologie (environnement, modèle de représentation des schémas, école, démarche et type d'intégration).
La méthodologie générale introduite sera réduite à un noyau minimal de fonctions sans lesquelles il ne serait pas possible de résoudre un problème d'intégration. Nous motiverons cette réduction par les raisons qui font que ces choix nous semblent judicieux.

Le sixième chapitre (IMPLEMENTATION DES FONCTIONS NECESSAIRES A UN ATELIER LOGICIEL: TRAMIS) décrira d'abord l'architecture de travail et les outils avec lesquels nous travaillerons (à savoir, l'environnement WINDOWS et l'atelier logiciel TRAMIS™). Nous donnerons ensuite une spécification détaillée des fonctions minimales et des algorithmes que nous avons réalisé[4].

---

[3] Par école nous entendons l'ensemble des personnes qui suivent une même ligne de pensée dans le processus d'intégration.

™ Cet atelier est développé conjointement par la société CONCIS de Paris(France) et les Facultés Universitaires Notre-Dame de la Paix de Namur (Belgique).

[4] En Annexe se trouve le listing du code source du module d'intégration que nous avons réalisé dans l'atelier logiciel TRAMIS.

Enfin, nous terminerons (Chapitre 7, CONCLUSION GENERALE) par une conclusion de notre travail avec les perspectives de prolongement de celui-ci dans l'avenir.

1.3 Historique du problème

Lorsque les premières bases de données sont apparues, c'était pour répondre au manque de flexibilité des fichiers: en effet, s'ils pouvaient parfois enregistrer des données de différents types, ils ne pouvaient gérer des relations entre ces données.
La base de données a permis de gérer ces liens et a offert des opérations supplémentaires sur les fichiers qu'elle manipulait (car une base de données est au départ une gestion intelligente de fichiers divers à laquelle on ajoute des outils de recherche, . . .).

A cette époque, les schémas conceptuels des données que contenaient les bases de données étaient petits et faciles à manipuler. Il n'y avait pas à proprement parler de problème d'intégration, tout était réalisable par un seul concepteur. Progressivement, comme le besoin de gérer des données de plus en plus complexes grandissait, il devint évident qu'un seul homme ne suffisait plus à la tâche.

Il a fallu partager la création des bases de données entre plusieurs groupes de concepteurs: les bases de données devenaient de plus en plus importantes au point de vue taille, profit, aspect critique.
Dans ce cas de figure, chacun développait son propre schéma local et lorsqu'il fallait passer à la mise en commun des résultats obtenus, de grandes difficultés pouvaient surgir à cause d'incohérences, d'incompatibilités, . . . dans les schémas à fusionner. La personne chargée de ce problème d'intégration devait procéder de manière entièrement manuelle.

De nos jours, la conception de bases de données est faite dans différents modèles (compliquant d'avantage la phase d'intégration). Une autre tendance qui voit le jour est celle d'intégrer des bases de données existantes (la probabilité qu'elles utilisent des modèles différents est encore plus grande: c'est le problème des bases de données hétérogènes).
Pour le résoudre, il y a peu de solutions et beaucoup de gens s'en sont rendus compte: de plus en plus de programmes de recherches sont lancés pour essayer de trouver des ressemblances dans un premier temps (pour avoir au plus vite un outil d'intégration qui s'avère de plus en plus indispensable) en attendant de formuler un fondement théorique.

L'évolution des recherches a été la suivante:

— les recherches, au départ, se concentraient sur la comparaison des objets de mêmes types (par exemple, les entités d'un schéma avec les entités d'un autre schéma décrit avec le même modèle) et généralement uniquement dans le modèle relationnel.

— Puis les gens se sont orientés vers un modèle sémantique du genre Entité/Association et les recherches sur l'intégration ont été très sommaires (du genre: seules les entités de même nom sont intégrées). Ce modèle est relativement riche mais offre peu de possibilités de comparaisons.

— Le pas suivant fut de travailler avec un modèle Entité/Association étendu qui était plus puissant pour représenter les données et surtout offrait des possibilités de faire des comparaisons pour l'intégration des objets semblables.

— Malgré tout, les recherches se limitaient à des objets de types semblables (même s'ils pouvaient être complexes). Pour enlever ces contraintes de comparaison d'objets de même nature, la recherche s'est orientée vers des modèles fonctionnels pour traduire des objets en des éléments comparables.

— Actuellement, les gens adoptent une approche plus flexible: "Chaque utilisateur doit pouvoir construire son propre sous-modèle, selon sa vue du monde réel" sans être "forcé d'accepter un sous-modèle d'un autre utilisateur . . . " *[Jardine 89]*. On cherche à préserver autant que possible les vues des différents concepteurs après la phase d'intégration.

1.4 Les deux approches

Le processus d'intégration de bases de données peut être vu selon deux approches:

— soit on intègre des schémas conceptuels (ou vues) lors de la création de la base de données,

— soit on intègre des bases de données qui existent déjà.

1.4.1 Intégration de vues

Cela consiste à prendre des vues de différentes bases de données (ou des vues partielles d'une base de données) construites séparément et de les fusionner pour obtenir un tout cohérent. C'est l'approche que l'on retrouve le plus souvent dans la littérature (*[Batini & Lenzerini 84]*, *[Spaccapietra & Parent 90]*, . . .) et l'optique que nous adopterons tout au long de ce mémoire. C'est pourquoi nous ne détaillons pas ce point de vue: il fera l'objet du Chapitre 5.

Cette opération n'est possible que lors de la création d'une base de données car lorsque les données sont présentes, les opérations à réaliser pour conformer les données au modèle intégré obtenu sont virtuellement impossibles ou en tout cas d'une complexité telle que personne ne s'est attaqué à ce problème (d'intégration de bases existantes) en suivant cette approche.

Les objectifs visés par cette approche sont:

- soit la conception en parallèle d'une grosse base de données,
- soit la centralisation de bases de données avec comme idée l'optimisation de la consommation des ressources (espace disque, . . .).

1.4.2 Intégration de bases de données existantes

Cette approche revient à détecter parmi les bases de données existantes, les parties communes pour obtenir une nouvelle base de données. Cependant, comme les données sont déjà présentes, il se peut qu'elles aient un formalisme différent. C'est pourquoi, il est pratiquement impossible de créer une base de données possédant des données semblables à celles des bases d'origine.

L'approche adoptée est de créer une interface qui puisse réduire les différences existant entre les différentes bases de données (quant à la structure de données "identiques" (par exemple une date représentée en lettre ou en chiffre), à la diversité des représentations possibles de la même réalité, . . .).

L'interface, pour être efficace, doit utiliser un modèle de représentation des données qui soit suffisamment riche pour pouvoir prendre en compte tous les cas qu'elle peut rencontrer. Elle doit en tout cas pouvoir manipuler des fonctions afin de traduire les données rencontrées en un modèle commun (celui de l'interface et qui est utilisé pour la présentation à l'utilisateur).

Cette interface complexe, peut être construite en plusieurs étapes (*[Motro 87]*) ou en une seule (*[Dayal & Hwang 84]*). Mais malgré tout, comme certaines données étaient présentes avant la création de cette interface supposée générale, il reste des cas où l'incompatibilité est trop grande pour pouvoir être intégrée. On a donc une base "incohérente" que l'interface essaie de cacher au mieux. Cette incohérence est impossible avec l'intégration de vues: les données sont introduites après l'obtention du schéma final (et doivent s'y conformer) mais le prix à payer, est de prévoir à l'avance tous les besoins à satisfaire[5].

L'objectif principal de cette approche est la centralisation des données (par exemple, dans l'optique d'un contrôle des données que l'on manipule). L'interface créée par le processus d'intégration dans ce cas offre une bonne vue d'ensemble.

[5] Dans les applications de petites tailles et aussi celles qui évoluent peu, c'est malgré tout réalisable.

CHAPITRE 2

ETAT DE L'ART SUR L'INTEGRATION DES VUES

2.1 Introduction

Il existe plusieurs méthodes d'élaboration des vues et de leur intégration avec le schéma global:

— on construit d'abord toutes les vues connues, on fait ensuite une intégration pour en déduire le schéma global *[Batini & Lenzerini 84]*,

— on construit d'abord le schéma global puis on dérive des vues de ce schéma global au fur et à mesure des besoins (approche relationnelle), ou

— on fait une conception parallèle des vues et du schéma global et on s'assure que l'ensemble se valide *[Tardieu & Al 79]*.

"La première approche qui présente plusieurs variantes, a l'avantage d'être modulaire et l'inconvénient de supposer toujours que les schémas externes sont le fait d'équipes de conception séparées et que leur intégration revient à en détecter les recouvrements. Or la plupart du temps, même pour les grosses applications, c'est la même équipe qui fait le schéma conceptuel, qui construit les schémas externes. Les problèmes se posent alors plus en terme de cohérence des contraintes d'intégrité entre elles.

La deuxième approche a l'avantage de la simplicité et l'inconvénient d'obliger les utilisateurs à se référer constamment à un expert unique (l'administrateur) qui maîtrise le schéma global, pour dériver leurs vues. L'administrateur en question aura donc une tâche continuelle de modélisation des besoins de chaque utilisateur. Ce qui peut avoir comme conséquence une certaine distorsion de la réalité parce qu'il appréhenderait ces besoins en fonction du schéma conceptuel dont il dispose". De plus, l'administrateur pourrait ne pas appréhender la totalité du problème et oublier (involontairement) certains aspects si le problème à traiter est de trop grande envergure.

"La troisième approche a l'avantage de supprimer les problèmes précédents et de valider les schémas par une double conception. Elle a l'inconvénient de la complexité des règles d'intégration. Mises à part quelques règles simples d'homonymie, de synonymie et d'inclusion de domaines et de contraintes, elle n'est envisagée que comme une tâche manuelle de confrontation entre les différents concepteurs" *[Comyn-Wattiau 90]*.

Nous allons présenter quelques démarches proposées par divers auteurs depuis 1979 jusqu'à nos jours. Toutes les démarches envisagées sont du premier type, à savoir intégration de schémas locaux en un schéma global[1], mais on retrouve certains aspects des autres techniques à des degrés divers dans chaque méthodologie.

[1] Le schéma global produit peut aller d'une "refonte" complete des vues d'origine a une "simple" interface qui adresse les requêtes aux bons endroits.

Dans le Chapitre 3, SYNTHESE DES PROBLEMES SUR L'INTEGRATION DES VUES, nous essayerons de déduire les éléments que nous retrouvons fréquemment dans chaque méthodologie et nous introduirons les concepts (rencontrés dans ces mêmes méthodologies) qui nous permettrons de faire une comparaison des diverses méthodes (voir Chapitre 4). De cela, nous établirons une méthodologie générale (voir Chapitre 5) qu'on pourrait appeler "idéale", qui essayera de reprendre les points les plus avantageux de chaque solution dans un certain contexte d'atelier logiciel.

2.2 La démarche de *[Elmasri & Wiederhold 79]*

C'est une des premières démarches d'intégration de vues. Les auteurs ont donc dû innover dans beaucoup (si pas toutes) d'étapes de l'intégration. Les problèmes abordés ne le sont pas à fond, tout n'est pas traité et la solution proposée est lourde et quasi inapplicable mais elle a le mérite d'exister.
Cette méthode a servi de point de départ de discussion à beaucoup d'autres démarches. Il suffit de consulter le nombre d'articles traitant de l'intégration de vues qui font référence à cet article.

L'objectif des auteurs est de fournir un modèle relationnel étendu qui puisse formaliser les dépendances fonctionnelles sous forme de tables relationnelles. Grâce à cela, on peut intégrer deux schémas tout en gardant la possibilité de modifier les schémas "d'origine".
En fait on ne garde des schémas d'origine qu'un sous schéma (éventuellement modifié) qui est compatible avec le schéma intégré. Ces hypothèses sont évidemment restrictives mais étaient un grand progrès à l'époque.

2.2.1 Le modèle de données

Les auteurs utilisent le modèle structurel qui est une extension du modèle relationnel de *[Codd 71]*. Ce modèle contient les éléments traditionnels du modèle relationnel, à savoir:

- les **attributs** qui sont des noms associés à des ensembles de valeurs appelés DOM(attribut),
- les **schémas de relation** qui sont des ensembles ordonnés d'attributs dont les éléments appartiennent au produit cartésien des différents attributs composant les schémas de relation,
- les **n-uplets** qui sont les éléments d'un schéma de relation, et
- les **relations** qui sont les ensembles des n-uplets des schémas de relation à un moment donné: ce sont des sous-ensembles des produits cartésiens correspondants.

Les extensions

Une des extensions de ce modèle est la notion de **connexion** entre deux schémas de relation X1 et X2.
Une connexion est établie par deux ensembles d'attributs Y1 et Y2 tels que:

— Y1 est un sous-ensemble de X1,

— Y2 est un sous-ensemble de X2, et

— les domaines de Y1 et Y2 sont égaux.

La connexion peut être généralisée à des ensembles d'attributs qui sont égaux à une fonction près (i.e. la troisième condition devient DOM(Y1) = f(DOM(Y2))).

Les auteurs utilisent plus particulièrement les deux connexions suivantes (voir Figure 2–1):

1. les connexions de **référence** qui sont établies quand Y2 est l'ensemble des attributs qui constituent l'identifiant de X2 et que Y1 est inclus dans l'ensemble des attributs identifiants (ou son complémentaire) de X1:
   — si Y1 est égal à l'ensemble des attributs identifiants de X1, la connexion de référence est dite **référence d'identité**,
   — sinon la connexion est dite **référence directe**.
   Les contraintes d'intégrité associées à ces connexions sont:
   i que chaque n-uplet de X1 doit être relié à un n-uplet de X2,
   ii que dans X2, seul les n-uplets non reliés à un n-uplet de X1 peuvent être supprimés;
2. les connexions d'**appartenance** qui sont établies quand Y1 est l'ensemble des attributs qui constituent l'identifiant de X1 et que Y2 est un sous-ensemble propre des attributs identifiants de X2.
   Les contraintes d'intégrité associées à ces connexions sont:
   i que chaque n-uplet de X2 doit être relié à un n-uplet de X1 (son propriétaire),
   ii que dans X1, chaque fois qu'on supprime un n-uplet, tous ceux dont il est propriétaire dans X2 doivent aussi être supprimés.

La seconde extension est la notion de **sous-relation** de R[X].
R'[Z] est cette sous-relation si:

— une référence identité existe de R' vers R,

— pour chaque n-uplet de R', il existe un n-uplet correspondant dans R tel que les identifiants de ces deux n-uplets sont égaux,

— à part les attributs qui forment l'identifiant, ces deux relations n'ont pas d'attributs en commun.

Dans ces conditions, on dit que R est la relation de base de la sous-relation R'.
Les contraintes d'intégrité des sous-relations sont:

i que chaque n-uplet de R' doit être connecté à un n-uplet de R,

ii que la suppression d'un n-uplet de R entraîne la suppression du n-uplet connecté dans R',

iii que si R' est une sous-relation de restriction (cfr infra la définition), chaque n-uplet de R qui appartient à la sous-relation doit aussi être présent dans la table R'.

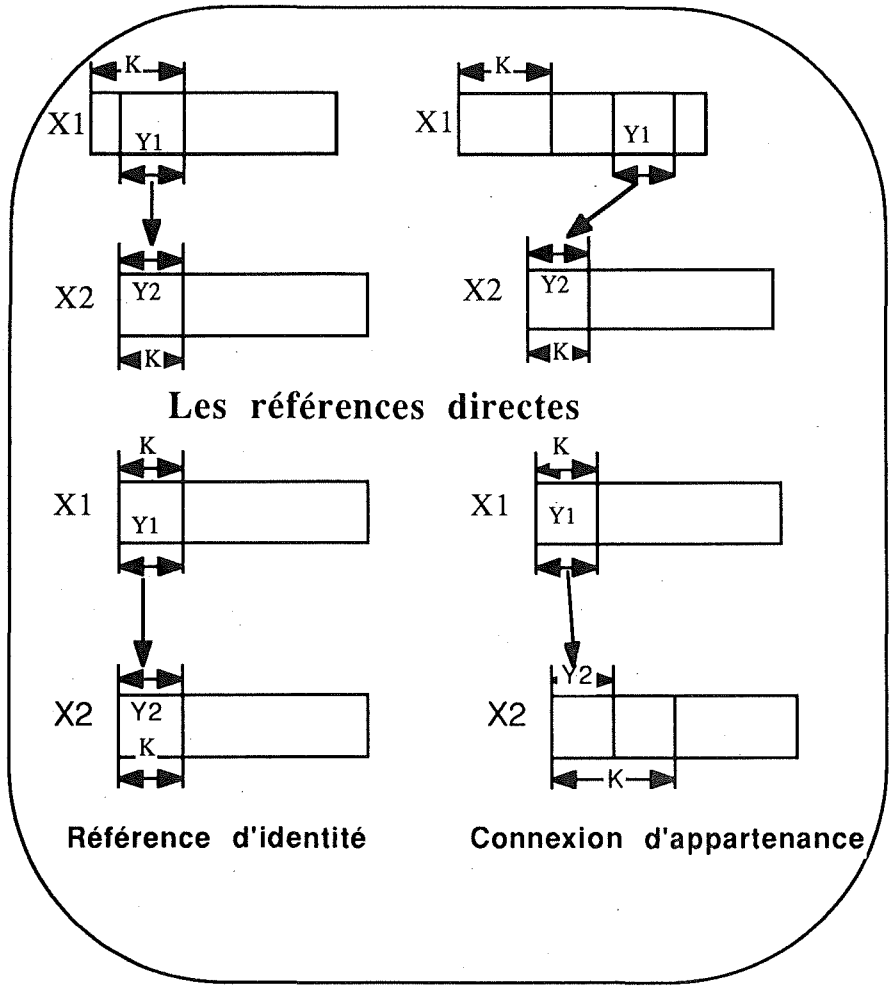
Types de relations

Les relations peuvent être utilisées pour représenter les objets du monde réel et les contraintes qui existent entre eux. Sur base de la définition suivante:

> une **relation entité** est une relation qui représente un type d'entités (dont les occurrences sont des objets du monde réel),

et suivant l'usage qui en est fait, *[Elmasri & Wiederhold 79]* font la classification suivante des relations qui peuvent apparaître dans le modèle structurel:

Figure 2–1: Types de connexions



- relation entité primaire: si aucune connexion "référence directe" ni connexion d'appartenance n'arrive sur cette relation,
- relation entité de référence: si elle est le point d'arrivée de connexions "référence directe",
- relation "imbriquée" (*nest reference [Elmasri & Wiederhold 79]*): si elle est le point d'arrivée d'une seule connexion d'appartenance,
- relation d'association: si elle est le point d'arrivée de plusieurs connexions d'appartenance: c'est ce qu'on utilise pour modéliser des associations N:N,
- relation lexicale entre les ensembles Y1 et Y2 d'attributs de R[X] pour définir une correspondance 1:1 si
  a. Y1 = les attributs identifiants de X,
  b. Y2 n'apparait nulle part ailleurs que dans R,
  c. Y1 et Y2 forment une partition de X,
  d. R est référencé par une ou plusieurs relation(s) dans le modèle de données.

Sur ces cinq types de relations, on peut définir des sous-relations. Ces sous-relations ne sont pas obligées d'avoir d'autres attributs que ceux de leur relation de base: dans ce cas on a une **sous-relation de restriction.**

2.2.2 La démarche

Les auteurs adoptent les définitions suivantes:

- le modèle de données est la représentation d'une vue nécessaire à un utilisateur ou une application,
- le modèle de base de données est la représentation du modèle créé par la fusion des différentes vues: si des conflits apparaissent en cours d'intégration, l'administrateur de bases de données doit intervenir pour modifier les schémas d'origine. Si malgré tout, il reste des incohérences, on abandonne ces parties,
- le sous-modèle de bases de données est la partie du modèle de données qui a pu être intégrée (parfois au prix d'une modification du schéma d'origine).

De ces définitions, on voit que l'application est fort restrictive (on ne peut intégrer toutes les vues et celles-ci peuvent être modifiées). Cela a été fait pour satisfaire les hypothèses choisies par les auteurs:

— possibilité de modifier les données (insertions / suppressions) des vues qui ont participé à l'intégration,

— limitation pour la présentation de la méthode à l'intégration de deux types d'entités reliés par une association: ces objets ont la même sémantique dans les deux vues à intégrer.

Si l'une des deux vues est plus générale, les auteurs décident de prendre la plus restrictive (et donc de prendre un sous-modèle de la vue la plus générale).

Les auteurs se limitent à deux vues et à deux entités car le modèle est riche et permet dès lors de modéliser la même situation par différentes représentations.
Il y a quatre possibilités de modéliser la même relation entre deux entités d'une vue (voir Figure 2–2):

a. une relation association entre les deux relations entités,

b. une référence directe,

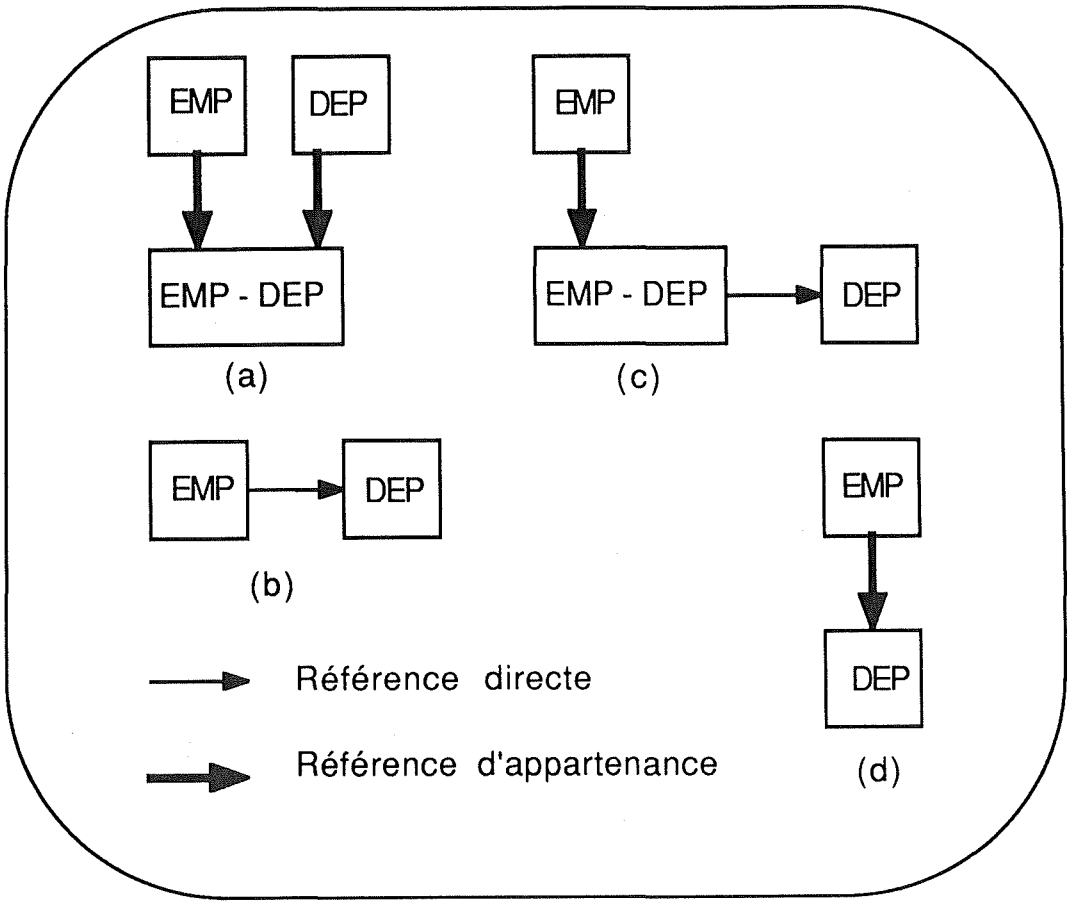
c. une imbrication de références, ou

d. une connexion d'appartenance.

Lorsqu'on introduit la deuxième vue, la combinatoire de ces hypothèses offrent déjà dix possibilités d'avoir à intégrer des structures différentes (à multiplier par deux car les relations ont une direction). Cependant certains cas sont symétriques et il ne reste en fait que douze cas à examiner.

Ces douze cas se trouvent dans l'analyse détaillée de *[Elmasri & Wiederhold 79]* avec schémas et explications. Ils ne seront cités ici qu'à titre d'indication. Ils ne recèlent en effet que peu d'enseignements sur la manière à suivre pour intégrer des vues: ils montrent cependant

Figure 2–2: Les quatre représentations possibles du réel



la complexité du problème de l'intégration sur un modèle pourtant simple et malgré tout épuré.
Ci-dessous les cas montrant les problèmes que posent les contraintes sous-jacentes aux modèles:

1. intégration d'une association et d'une imbrication de références,
2. intégration d'une association et d'une référence directe,
3. intégration d'une association et d'une référence imbriquée ("nest reference"),
4. intégration de deux imbrications de références,
5. intégration d'une imbrication de références et d'une référence directe de A vers B,
6. intégration d'une imbrication de références et d'une référence directe de B vers A,
7. intégration d'une imbrication de références et d'une référence imbriquée de A vers B,
8. intégration d'une imbrication de références et d'une référence imbriquée de B vers A,
9. intégration de deux références directes
10. intégration d'une référence directe et d'une référence imbriquée de A vers B,
11. intégration d'une référence directe et d'une référence imbriquée de B vers A, et

12. intégration de deux références imbriquées.

Pour intégrer deux vues dont les données de l'une sont incluses dans l'autre, les auteurs suggèrent de mettre les attributs communs en relation de base et de faire deux sous-relations avec les attributs restants. Les données introduites dans une des vues d'origine seront cachées dans la vue intégrée tant qu'elles ne seront pas à leur tour introduites dans la seconde vue.

2.3 La démarche de *[Yao & Al 82]*

Les auteurs ont voulu fournir une aide à la conception de grosses bases de données et à la transformation des transactions accompagnant les schémas qu'on désire intégrer.

2.3.1 Le modèle de données

C'est un modèle de données fonctionnel que les auteurs appellent FDM (*F*unctional *D*ata *M*odel). Il possède deux concepts principaux:

- les **noeuds**, qui peuvent être simples ou des n-uplets.
  Un noeud simple représente un ensemble de données atomiques (des attributs). Il possède
  - — un nom,
  - — un type de données (caractère, entier ou réel), et
  - — une longueur (en byte).
  - — Il est associé à un domaine (avec un nom) dans lequel sont prises les valeurs du noeud simple.

  Le noeud n-uplet représente des relations non fonctionnelles entre noeuds. Il contient un ensemble de n-uplets de données atomiques. Il possède
  - — un nom,
  - — les noms des éléments participant au n-uplet.
- les **fonctions**, qui peuvent être 1:N, 1:1 ou identité (de valeurs dans des noeuds distincts). Les fonctions peuvent être aussi totales ou partielles.

Le modèle est complété par des assertions (contraintes d'intégrité) qui stipulent les contraintes qu'on ne peut exprimer à l'aide de ces deux concepts. Une telle nécessité apparaît par exemple lorsqu'on a plusieurs fonctions partielles au départ d'un même noeud: si on veut modéliser qu'un élément du noeud est la source d'une fonction, on ne peut le faire qu'au moyen de contraintes disant que l'intersection des domaines de définitions de ces deux fonctions n'ont pas d'intersection.

2.3.2 La démarche

Le principe est de regrouper toutes les vues utilisateurs, puis de simplifier le résultat en éliminant les fonctions et les noeuds redondants. Il faut fusionner les noeuds avec les mêmes valeurs, puis ceux qui sont sous-ensembles d'autres noeuds. Il faut enlever les fonctions redondantes et adapter les transactions. L'adaptation est faite automatiquement par les opérateurs de restructuration mis à la disposition de l'utilisateur.

C'est l'utilisateur qui opère l'intégration mais il est aidé par un programme qui possède des heuristiques basées sur le modèle FDM afin de poser un minimum de questions à l'utilisateur.

Le résultat obtenu est un schéma global avec les transactions écrites en termes du schéma global mais les vues d'origine sont complètement perdues.

Le langage des transactions

Le langage TASL (*Tr*Ansaction *S*pecification *L*angage) utilise trois opérateurs d'accès aux données:

- **ENTRY (A) WHERE P** retourne l'ensemble des valeurs de A qui vérifient le prédicat P,
- **ITEM (F,A) WHERE P** retourne l'ensemble des valeurs de F(A) qui vérifient le prédicat P, et
- **SET (F,B) WHERE P** retourne l'ensemble des valeurs que donne l'inverse de F appliquée à B et qui vérifient le prédicat P.

Le langage TASL possède aussi des opérateurs algébriques sur les noeuds:

- intersection,
- union, et
- différence.

Il possède enfin les différents types de commandes:

- affectation de variables,
- boucles,
- instructions conditionnelles,
- instructions d'entrée / sortie,
- opérations arithmétiques, et
- manipulation des noeuds (ajout d'occurrences, . . .).

A chaque ordre de restructuration des schémas, les transactions associées sont mises à jour.

Les opérateurs de restructuration

Il n'y a que deux opérateurs:

- REMOVE (F, F1, ..., Fn) enlève la fonction redondante F et remplace partout dans les algorithmes F(x) par $F_n(...F_1(x)...)$. (Exemple 2–1),
- MERGE (A, B, C) fusionne les noeuds A et B en C qui est l'union des deux.
  Toutes les fonctions où A ou B participaient sont remplacées par des fonctions utilisant C à la place. L'opérateur MERGE doit introduire un noeud de séparation, lorsqu'on veut préserver la distinction d'éléments, si A est inclus dans B (Exemple 2–2).

Exemple 2–1: L'opérateur REMOVE

On dit que la fonction quelconque (1:N, 1:1 ou identité) F définie entre les noeuds B et C (notée F: B ⇒ C) est *redondante* avec les fonctions $F_1 : A_0 \Rightarrow A_1; \ldots ; F_n : A_{n-1} \Rightarrow A_n$ si B réduit au domaine de définition de F (noté B.F) $\subseteq A_0.F_1$ et si $\forall a \in B.F, F(a) = F_n(...F_2(F_1(a))...)$.
Si les fonctions identités $I_1 : B \Leftrightarrow A_0$ et $I_2 : C \Leftrightarrow A_n$ représentent les inclusions de B.F dans $A_0.F_1$ et de C.F dans $A_n.F_n$, alors chaque occurence du code TASL

```
SET (F, argument) WHERE P
```

sera remplacée par la nouvelle expression

```
ITEM (I1, SET (F1, ..., SET (Fn, ITEM (I2,argument))  . . . ) WHERE P)
```

Exemple 2–2: L'opérateur MERGE

Si on veut fusionner A et B quand A est inclus dans B, l'opérateur *MERGE (A, B, C)* va créer un noeud de séparation SAB et une dépendance S: C ⟶ SAB. Chaque occurence TASL de

```
ENTRY (A) WHERE P
```

sera remplacée par

```
SET (S, ENTRY (SAB) WHERE SAB = 'A') WHERE P
```

2.4 La démarche de *[Batini & Lenzerini 84]*

Les auteurs donnent une démarche d'aide à l'intégration des vues. Le but de cette démarche est d'essayer de résoudre les trois principaux problèmes qui peuvent survenir lorsqu'on veut intégrer des vues.

Ces trois problèmes sont les suivants:

1. le manque de rigueur du modèle,
2. les perceptions multiples pour un même objet, et
3. le manque de fiabilité.

Le premier problème est dû au fait que plusieurs concepteurs peuvent représenter le même univers du discours de différentes façons.
En ce qui concerne le second problème, c'est la possibilité d'avoir dans plusieurs schémas à intégrer des synonymes (les mêmes objets d'une classe de l'univers du discours sont représentés par des noms différents) et homonymes (les classes d'objets différentes ont le même nom).
Enfin le dernier problème, est dû au fait que le concepteur peut faire des choix erronés pour les noms et propriétés des objets de la vue.

2.4.1 Le modèle de données

Le modèle utilisé est le modèle Entité/Association de *[Chen 76]* étendu.

Le modèle possède trois concepts de base:

1. l'entité,
2. l'association,
3. attribut.

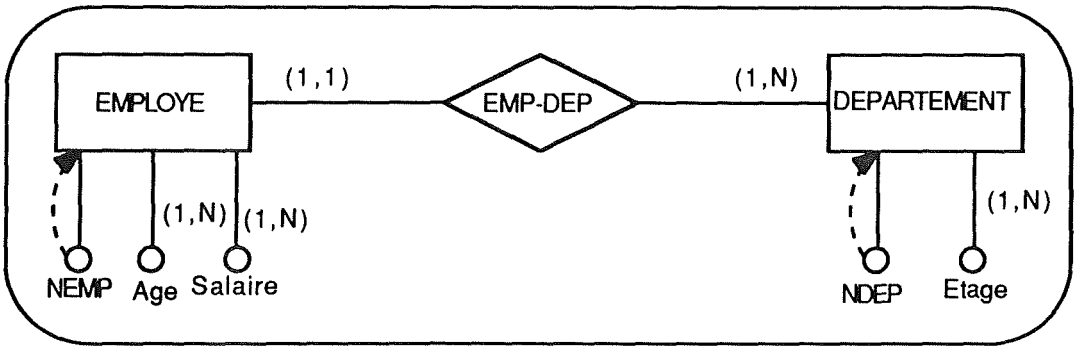
Le modèle de *[Chen 76]* est étendu au mécanisme de généralisation/spécialisation.

les possibilités offertes par ce modèle sont les suivantes:

- à chaque concept de base est associé un nom et un ensemble de synonymes (éventuellement vide);
- au concept "ATTRIBUT" est attaché un type de valeurs. Les types autorisés sont:
  - entier,
  - booléen,
  - chaîne de caractères, ou
  - défini par l'utilisateur;
- à chaque concept "ENTITE" est associé un ou plusieurs identifiants. L'identifiant est un ensemble non vide d'attributs et/ou d'entités;
- on définit les cardinalités minimales (Card-min) et maximales (Card-max) pour:
  - les attributs attachés à une entité ou une association, et
  - les entités jouant un rôle dans une association;
- on peut définir des dépendances fonctionnelles entre attributs d'une même entité ou d'une même association.

La Figure 2–3 montre un exemple de schéma Entité/Association. La notation graphique est la suivante: le rectangle représente une entité, le losange représente une association, le cercle représente un attribut et la courbe en pointillé indique l'identifiant de l'entité.

Figure 2–3: Exemple de schéma Entité/Association



Ajouté à cela, on fait la distinction entre les propriétés et les contraintes des différents concepts de base du modèle (voir Tableau 2–1).

Tableau 2–1: Les caractéristiques des concepts de base du modèle E/A

Concept	Propriétés	contraintes
Entité	Attributs, association, généralisation/spécialisation	Card-min et card-max de l'entité participante aux associations, identifiants
Association	Entités, attributs	Card-min et card-max des entités impliquées
Attribut	Entité/Association	Type, identifiants, card-min, card-max

2.4.2 La démarche

La stratégie d'intégration choisie par les auteurs est l'intégration binaire (en échelle). Partant de n schémas auxquels le concepteur a assigné des poids[2], les auteurs les intègrent deux à deux, en commençant par le schéma de poids supérieur. Ce choix a pour but de simplifier le processus d'intégration. On peut concevoir que l'intégration de n schémas à une complexité de l'ordre du n^2.
Le processus d'intégration de deux schémas est découpé en trois phases:

1. l'analyse des conflits,
2. la fusion des schémas, et
3. la restructuration du schéma intégré.

Analyse des conflits

Le but de cette phase est de détecter et résoudre toutes les incohérences qui existent dans la représentation des objets des deux schémas à intégrer. Cette phase résout les trois problèmes (manque de rigueur, perceptions multiples et manque de fiabilité) énoncés plus haut en interaction avec le concepteur. C'est le concepteur qui prend les décisions et ce en fonction d'un certain nombre d'indications faites par l'outil.

Cette phase est subdivisée en deux parties: on résout d'abord les problèmes de terminologie (synonymes, homonymes) et une fois cette étape finie, on passe à la vérification de la compatibilité de la modélisation qui consiste à résoudre le cas où un objet n'a pas le même type dans les deux schémas (attribut dans l'un, entité dans l'autre par exemple).

Analyse des noms Cette analyse permet de détecter d'éventuels synonymes et homonymes. La détection de conflits de noms se fait par des indications. Le concepteur analyse celles-ci et résout les différents problèmes manuellement.

Les indications prises en considération par les auteurs sont les suivantes:

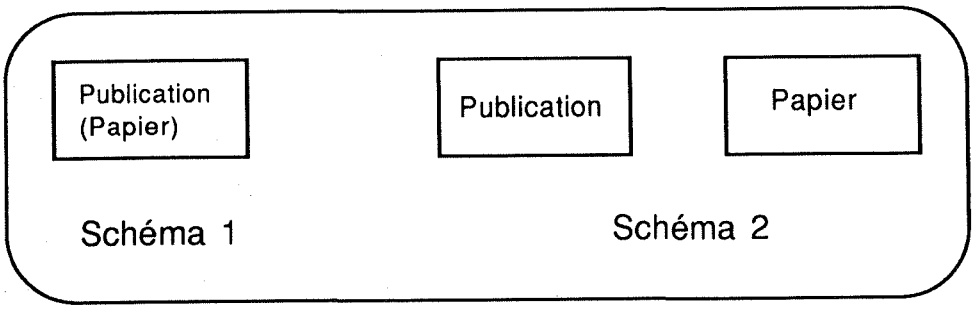
- l'anomalie dans les noms,
- la ressemblance entre objets, et
- la différence entre objets.

La première indication est découverte par l'inspection des noms et synonymes des différents objets. Il se peut que deux noms/synonymes A(B) soient attachés au même objet dans le schéma S1 et à des objets différents A et B dans le schéma S2. Cette situation marque l'existence de rapports d'homonymie et de synonymie entre les objets des deux schémas. Dans un tel cas, un moyen de résoudre ce problème est de définir une propriété

[2] Ce poids est la mesure de qualité et/ou des informations pertinentes à l'intérieur de chaque schéma à intégrer.

inter-schémas[3] de sous-type entre B et A. Cette propriété sera traitée dans la phase de restructuration. La Figure 2–4 montre un exemple d'anomalie de noms. Ce problème peut être résolu en assignant le nom unique "Papier" à l'entité du schéma 1 + la définition d'une propriété inter-schémas de sous-type entre l'entité "Papier" et l'entité "Publication" dans le schéma 2.

Figure 2–4: Exemple d'anomalie de noms



L'indication de ressemblance est une indication de synonymie entre deux objets: objets de noms différents et possédant certaines caractéristiques communes. Tandis que la présence de différences est une indication d'homonymie entre deux objets: objets de même nom et ayant un certain nombre de caractéristiques différentes.

Pour rechercher les synonymes et homonymes parmi les objets des deux schémas et afin d'éviter une recherche trop longue, les auteurs se limitent aux comparaisons suivantes: si les deux objets comparés sont:

- de même type, on compare
  - — le type de valeurs pour les attributs,
  - — les associations et attributs pour les entités,
  - — les attributs et les entités jouant un rôle dans l'association pour les associations;
- de types différents,
  - — dans le cas entité/attribut, on compare le type de l'attribut et le type des identifiants de l'entité, ou
  - — dans le cas entité/association, on compare les attributs et le type des identifiants.

Compatibilité de la modélisation L'objectif de cette étape est d'attacher le même type à tous les objets qui représentent la même classe d'objets (objets qui ont le même nom) et qui sont de types différents dans les deux schémas. Cette étape a pour but aussi d'essayer de détecter les choix erronés que les concepteurs ont pu faire.

[3] C'est une propriété qui apparaît seulement au moment de l'intégration de schémas entre eux.

L'analyse se concentre sur les objets de même nom et de types différents. On dit alors que les objets ont des types incohérents.
Pour résoudre cette incohérence, les auteurs donnent des transformations qui préservent l'invariance du contenu de l'information dans les deux schémas. Ces transformations ne sont valables que pour les concepts atomiques: entité, association, attribut.
Les transformations possibles sont les suivantes:

i incohérence de types entre un attribut d'une entité et une entité: deux transformations différentes sont définies:
   - transformation T1: transformer l'attribut en une entité. Un attribut peut toujours être transformé en une entité.
   - transformation T2: transformer l'entité en un attribut. Cette transformation est possible seulement si un certain nombre d'hypothèses[4] sur les propriétés de l'entité sont respectées.

ii incohérence de types entre une entité et une association: deux transformations alternatives sont définies:
   - transformation T3: transformer l'association en une entité. Une association peut toujours être transformée en une entité.
   - transformation T4: transformer l'entité en une association. Cette transformation est réalisable uniquement si un certain nombre d'hypothèses[4] sur les propriétés de l'entité sont respectées.

iii incohérence de types entre un attribut d'une association et une entité: dans ce cas la seule transformation est:
   - transformation T5: transformer un attribut en une entité. L'attribut d'une association peut toujours être transformé en une entité de la façon suivante:
     1. l'association est transformée en une entité (transformation T3).
     2. l'attribut est maintenant un attribut de l'entité et on le transforme en une entité (transformation T1).

La dernière étape de cette phase s'occupe de la vérification de la compatibilité des objets représentant la même classe d'objets de l'univers du discours. On dit que deux objets sont:

- IDENTIQUES s'ils ont le même nom, le même type et ont les mêmes propriétés et contraintes;
- COMPATIBLES s'ils ont le même nom, le même type et ont des propriétés et contraintes non contradictoires entre elles;
- INCOMPATIBLES s'ils ont le même nom, le même type et ont au moins une propriété et/ou contrainte incompatible.

Durant la phase de fusion les objets identiques ou compatibles seront simplement superposés l'un dans l'autre. En ce qui concerne les objets incompatibles, le concepteur doit résoudre le problème avant de passer à la phase suivante.

[4] Nous renvoyons le lecteur à l'article *[Batini & Lenzerini 84]* pour le détail de ces différentes hypothèses.

Les incompatibilités entre les objets (ayant le même nom et le même type) des deux schémas peuvent être:

- la différence de cardinalités pour le même attribut ou l'entité (jouant un rôle de l'association),
- l'identifiant dans un schéma ne l'est pas dans l'autre schéma,
- une entité est (transitivement) un sous-type d'une autre entité dans un schéma et on a l'inverse dans l'autre schéma, et
- des dépendances fonctionnelles différentes sont définies pour les mêmes attributs d'une entité dans les deux schémas.

les solutions possibles pour l'incompatibilité sont les suivantes:

- une représentation est choisie par rapport à l'autre, ou
- une représentation commune est construite telle que toutes les contraintes des deux schémas sont reprises dans le schéma intégré.

La première solution est prise en considération quand une analyse détaillée est possible pour voir laquelle des deux représentations est la plus fiable. Tandis que la deuxième solution est appliquée quand les deux représentations sont sur le même pied d'égalité en fonction des critères de fiabilité et de compréhension.

Fusion des schémas

A ce moment de l'intégration tous les conflits pour la représentation des objets dans les deux schémas ont été résolus.

La fusion des schémas est une simple superposition des deux schémas avec une coloration différente des objets selon qu'ils sont propres au schéma 1, propres au schéma 2 ou communs aux deux schémas.

Enrichissement et Restructuration

Cette phase de l'intégration est composée de trois tâches essentielles à résoudre:

1. l'analyse des propriétés inter-schémas,
2. l'analyse des chemins redondants, et
3. la restructuration du schéma.

Analyse des propriétés inter-schémas Ces propriétés ont été découvertes lors des étapes précédentes, et dans ce cas elles sont simplement superposées dans le schéma intégré.

Des propriétés nouvelles peuvent être trouvées au moyen d'une analyse en profondeur sur le schéma intégré. Seuls les objets de différentes couleurs sont analysés.

Analyse des chemins redondants Après la phase de fusion, il se peut que des cycles redondants dans le schéma intégré apparaissent. On distingue trois types de cycles:

- ceux où aucune association ne peut être éliminée sans perte d'information,
- ceux où seulement une seule association peut être éliminée, ou bien
- ceux où plus de deux associations peuvent être éliminées.

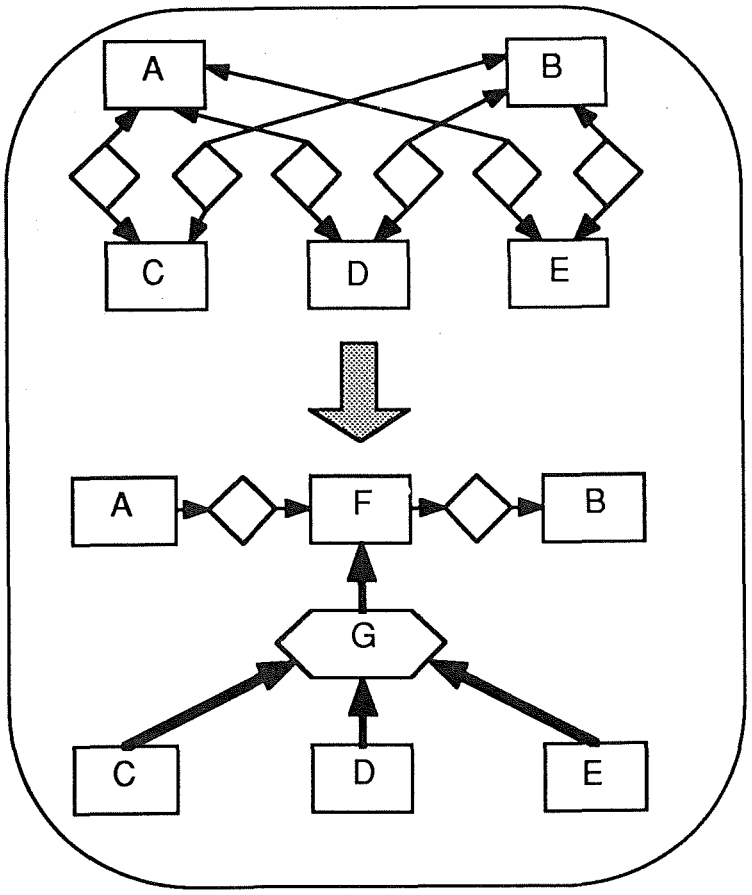
Seul le deuxième cas offre la possibilité de restructurer le schéma par la suppression d'une association. Par contre dans le premier cas, on perd de l'information, alors que dans le dernier cas, il faut trouver le chemin le plus performant possible au niveau de la conception physique du schéma intégré.

Restructuration du schéma L'objectif de cette étape est d'analyser en détail le schéma intégré dans le but:

- d'avoir un schéma intégré compréhensible, et
- de gagner en clarté et en simplicité dans la représentation de l'univers du discours.

L'exemple de la Figure 2–5 montre un cas où il est possible de gagner en clarté et en compréhension en restructurant le schéma intégré à l'aide du mécanisme de généralisation/spécialisation.

Figure 2–5: Restructuration du schéma intégré



La démarche proposée par *[Batini & Lenzerini 84]* est applicable uniquement lors de l'analyse conceptuelle des schémas. Elle est entièrement interactive et laisse la prise de décision au concepteur.

2.5 La démarche de *[Dayal & Hwang 84]*

La démarche proposée est une intégration par généralisation des bases de données existantes. Le schéma global est un schéma virtuel servant à résoudre toutes les incohérences (sur la structure et sur les données) entre les données existantes dans les différentes bases de données.

[Dayal & Hwang 84] décomposent le problème de l'intégration en trois sous-problèmes, à savoir:

1. l'intégration de deux schémas de bases de données: elle consiste en l'étude des différences de structuration des objets,
2. l'intégration de deux bases de données qui peuvent contenir des données mutuellement inconsistantes, et
3. la transformation de requêtes portant sur le schéma global en une suite de sous-requêtes dans les différents langages utilisés par les schémas locaux.

2.5.1 Le modèle de données

Les auteurs utilisent un modèle fonctionnel: FDM (**F**unctional **D**ata **M**odel), qu'ils ont étendu pour que celui-ci permette d'exprimer les généralisations. Ils présentent d'autre part un outil de traduction des requêtes[5] qui transforme une question portant sur la vue globale en une suite de sous-questions portant sur les vues locales.

Le modèle fonctionnel de données possède deux concepts de base:

- l'entité: pour représenter l'univers du discours des objets, et
- la fonction: pour représenter les propriétés des objets et les associations entre les objets. Les fonctions peuvent être monovaluées ou multivaluées.

Le modèle est étendu au mécanisme de généralisation, dans le but d'apporter une plus grande flexibilité dans la modélisation de la sémantique des schémas. Ce mécanisme permet d'éviter les attributs facultatifs dans le schéma global.

Le modèle fonctionnel FDM se représente très simplement en utilisant les conventions graphiques de la Figure 2–6.

La Figure 2–7 illustre un exemple de schéma exprimé dans ce modèle.

[5] Cet outil est une variante de NQUEL, qui est un langage non procédural de requêtes.

Figure 2–6: Les principaux concepts du modèle FDM et leur représentation graphique

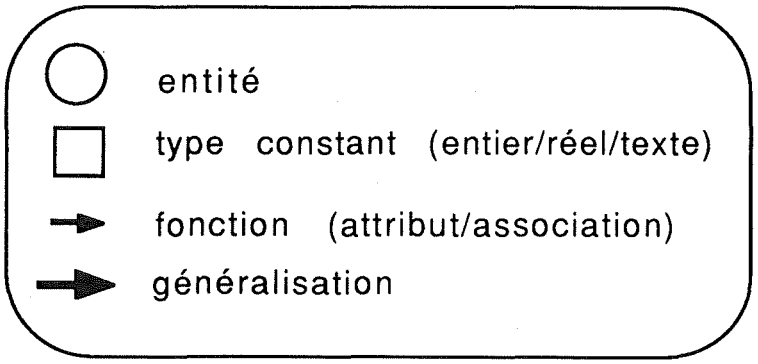
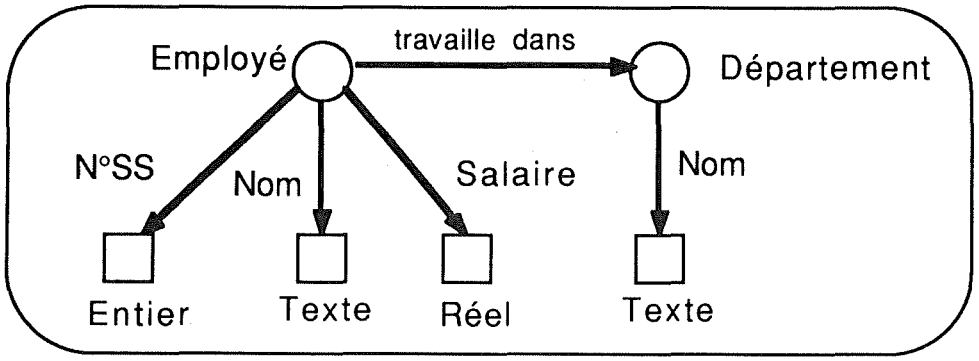


Figure 2–7: Exemple de schéma fonctionnel



2.5.2 La démarche

L'approche choisie par *[Dayal & Hwang 84]* est de préserver toutes les informations dans les schémas locaux. Elle consiste à faire l'intégration en deux étapes. La première résout les différences entre les schémas de telle façon que les entités dans tous les schémas soient semblables. Après quoi, on réalise la combinaison des vues par généralisation. Les différences de données sont résolues en définissant des fonctions sur le "supertype".

Intégration des deux schémas des bases de données

L'intégration de schémas comprend la résolution:

- des conflits de noms,
- des différences d'échelles,
- des différences structurelles, et
- des différences d'abstractions.

Les conflits de noms Les conflits de noms nécessitent un renommage. Si des entités (ou des fonctions) représentant le même univers du discours de l'objet sont présentes avec des noms différents (synonymes) dans différents schémas, alors on leur donne le même nom. Par contre, si des entités (ou des fonctions) représentent des objets différents dans les schémas, avec le même nom (homonymes), on résout le conflit en donnant des noms différents dans le schéma global.

Les différences d'échelles Les mêmes fonctions peuvent être stockées dans différents schémas locaux avec des échelles différentes. Par exemple l'attribut "HAUTEUR" est mesuré en pouce dans un schéma et en centimètre dans un autre schéma. Cette différence d'échelles ou d'unités pour les valeurs des attributs est résolue à l'aide de fonctions de conversion. Ces fonctions de conversion sont stockées dans une base de données auxiliaire (au schéma global).

Les différences structurelles On entend par différences structurelles: les entités qui n'ont pas exactement les mêmes attributs ou les associations qui ne sont pas modélisées de la même façon. Ce problème peut être résolu par des fonctions de généralisation.

La technique conseillée par les auteurs est lorsqu'on intègre des schémas avec des entités et des fonctions différentes, on définit d'abord des fonctions et des entités virtuelles. Après on utilise le mécanisme de généralisation. Une possibilité de résolution de ce problème est illustrée à la Figure 2–8.

Les différences d'abstractions Les entités peuvent être définies à différents niveaux de généralisation dans les schémas locaux. Le chemin naturel pour intégrer ces schémas est via le mécanisme de généralisation.

Intégration des deux bases de données

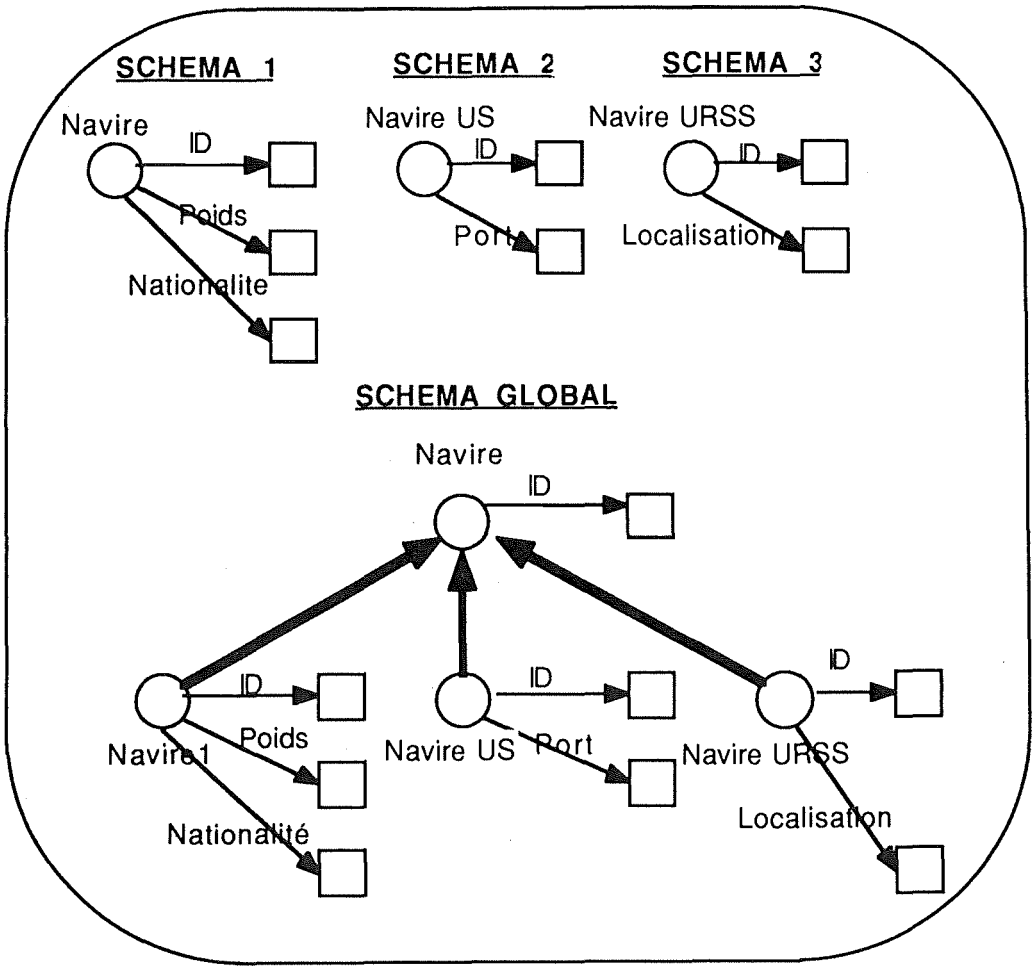
Une fois que la structure du schéma global virtuel est définie, il reste à résoudre les conflits entre les données des schémas locaux. Il existe deux sortes de conflits:

- les données dans les schémas sont mutuellement inconsistantes parce que certaines sont incorrectes,
- les données dans les schémas sont mutuellement inconsistantes mais correctes.

Le seul moyen pour résoudre le premier conflit, est de garder les données les plus crédibles. Par contre, pour résoudre le second problème, on peut avoir les trois cas suivants:

- les identifiants ne sont plus uniques quand on réunit les bases, on ajoute alors le nom de la base comme composant de l'identifiant,
- certains attributs ont des valeurs locales, on ajoute le nom de la base pour permettre l'identification des valeurs,
- certaines données sont obsolètes, on ajoute la date comme complément d'identifiant.

Figure 2–8: Exemple d'intégration par l'emploi du mécanisme de généralisation



Modification des requêtes

Ils proposent un algorithme de traduction d'une question portant sur une vue globale en sous-questions portant sur les vues locales ayant concouru à l'intégration. La particularité de la modification de questions est liée au fait que les vues peuvent être définies par généralisation. Si la question comporte des fonctions définies sur une entité obtenue par généralisation, l'outil calcule la meilleure partition de cette entité.

Par exemple, on veut construire la relation RICHE qui contiendra le nom de tous les employés dont le salaire est supérieur à 500 000. EMPLOYE est défini par généralisation de EMP1 et EMP2, qui sont les deux relations EMPLOYES de deux sites différents d'une entreprise. Le nom EMPLOYE est celui qui figure dans EMP1 ou dans EMP2 ou éventuellement dans les deux. Par contre le calcul de l'attribut salaire de EMPLOYE diffère selon que l'employé considéré travaille dans un des deux sites de l'entreprise ou au contraire dans les deux sites: dans ce dernier cas, son salaire résultant est la somme des deux salaires stockés dans les relations initiales:

```
Nom(e) = Nom1(e) si e appartient à EMP1
Nom(e) = Nom2(e) si e appartient à EMP2 - EMP1
Salaire(e) = Sal1(e) si e appartient à EMP1 - EMP2
Salaire(e) = sal2(e) si e appartient à EMP2 - EMP1
Salaire(e) = sal1(e) + sal2(e)
                     si e appartient à l'intersection
                     de EMP1 et EMP2
```

L'utilisateur ayant accès au schéma global formulera la question de la façon suivante:

```
RANGE OF e IS EMPLOYE
RETRIEVE INTO RICHE (Nom := Nom(e))
WHERE Salaire(e) > 500000
```

La question sera transformée en trois sous-questions portant sur EMP1 - EMP2, EMP2 - EMP1 et EMP1∩ EMP2; puis on établit l'union des trois résultats:

```
1. RANGE OF e IS EMP1 - EMP2
   RETRIEVE INTO temp1 (Nom := Nom1(e))
   WHERE Sal1(e) > 500000

2. RANGE OF e IS EMP2 - EMP1
   RETRIEVE INTO temp2 (Nom := Nom2(e))
   WHERE Sal2(e) > 500000

3. RANGE OF e IS intersection (EMP1,EMP2)
   RETRIEVE INTO temp3 (Nom := Nom1(e))
   WHERE (Sal1(e) + Sal2(e)) > 500000
```

Finalement, RICHE = temp1 ∪ temp2 ∪ temp3

2.6 La démarche de *[Biskup & Convent 86]*

Les auteurs ont voulu donner un cadre théorique au processus d'intégration de vues inspiré des techniques qui les ont précédés (tout ce qui a été développé avant 86). Ils ont surtout approfondi et formalisé:

- le modèle de données,
- un langage formel de spécification des connexions entre les vues,
- la notion de conflit,
- la notion d'inclusion de vue dans un schéma global (pour garantir la complétude et la préservation de la sémantique des vues utilisateurs dans le schéma intégré), et
- une méthode formelle d'intégration.

De ce cadre théorique, ils ont eux-mêmes déduits une méthode d'intégration qui s'insère dans le développement de bases de données, qui pour les auteurs contient les phases:

1. d'analyse des besoins (spécifiés formellement),
2. de modélisation des vues (dans un modèle abstrait et conceptuel),
3. d'intégration des vues (fournissant un schéma général dans le même modèle abstrait),
4. de conception logique, et
5. de conception physique.

2.6.1 Le modèle de données

Les auteurs utilisent le modèle relationnel de *[Codd 71]*. Ils adoptent cependant certaines restrictions, qu'ils appellent normalisation du schéma et qui selon eux, si elles semblent fort techniques, n'en sont pas moins souvent vérifiées dans la réalité. Voici ces contraintes qu'il faudra veiller à respecter lors de la modélisation des vues:

a. les seules dépendances fonctionnelles admises dans les schémas de relations sont celles qui existent entre les attributs qui forment l'identifiant et les autres;

b. les contraintes d'inclusion ne sont valables
   — qu'entre tables distinctes,
   — les attributs (qui sont mentionnés dans la contrainte d'inclusion) d'une des deux tables doivent être ceux qui constituent l'identifiant, et
   — les attributs de l'autre table doivent être soit un sous-ensemble des attributs identifiants ou un sous-ensemble des attributs non identifiants (mais pas un mélange des deux);

c. les contraintes d'exclusion ne sont valables
   — qu'entre tables distinctes, et
   — les attributs (qui sont mentionnés dans la contrainte d'exclusion) des deux tables doivent être ceux qui constituent l'identifiant.

A ce modèle, les auteurs ajoutent des contraintes d'intégration, modélisées comme les contraintes d'intégrité. Une base de données étendue est l'ensemble des relations, contraintes d'intégrité et contraintes d'intégration (qui peuvent ne pas exister i.e. le troisième ensemble est vide). Les contraintes (exprimées au niveau de la définition (intension) de la relation et portant sur les n-uplets (extension) de la dite relation) sont les suivantes:

- **identité** entre deux relations de vues différentes: les attributs (mentionnés dans la contrainte) de chaque relation contiennent au moins les attributs identifiants et leurs domaines sont égaux;
- **sélection** entre deux relations de vues différentes: mêmes conditions. Cela correspond à une identité entre la première relation et toutes les occurrences de la deuxième qui vérifient un prédicat;
- **disjonction** entre deux relations de vues différentes: les attributs constituent l'identifiant et leurs domaines sont égaux;
- **contenance** entre deux relations de vues différentes: elle s'applique soit dans les mêmes conditions que l'identité, soit les mêmes conditions que la contrainte d'inclusion.

2.6.2 La démarche

La démarche permet d'intégrer plusieurs vues simultanément et garantit la préservation des contraintes d'origine.

On part de la définition des vues (relations + contraintes obtenues par la modélisation) et des contraintes d'intégration ajoutées après analyse des vues à intégrer.

On combine les vues en un schéma sans conflit avec les vues initiales (ce qu'on peut prouver en projetant le schéma global sur chaque vue qu'on veut intégrer: si la projection et la vue sont identiques, et ce pour chaque vue, les auteurs définissent cet état de fait comme des vues sans conflit). Les auteurs supposent que si ce n'est pas possible, c'est qu'une des vues a été mal spécifiée.

Ce résultat peut être fortement redondant: il faut supprimer les contraintes d'intégration une à une (ce qui est fait par un algorithme) pour obtenir un schéma qui respecte le modèle relationnel normalisé, qui est équivalent au schéma combiné (deux schémas sont équivalents s'ils sont inclus l'un dans l'autre et l'inclusion de V dans G est définie par l'existence d'une fonction qui appliquée à l'ensemble de toutes les occurrences valides de G donne l'ensemble de toutes les occurrences valides de bases de données respectant le schéma V).
Le résultat est complété par une fonction qui assure le mapping entre le schéma combiné et le schéma optimisé (il y en a au moins une vu que les deux schémas sont équivalents).
Les requêtes sont elles-mêmes optimisées en composant d'abord la requête avec cette fonction de mapping pour accéder directement au schéma global et obtenir ainsi plus rapidement la réponse.

L'intégration proprement dite

Principes: On part du schéma combiné et de la fonction identité et on veut arriver à un schéma sans plus de contrainte d'intégration qui soit aussi conforme au schéma normalisé.

- Les contraintes d'identité sont éliminées en faisant l'union des attributs des deux relations pour en obtenir une nouvelle et enlever les deux d'origine;
- les contraintes de sélection sont éliminées en enlevant la relation qui est contenue dans l'autre, elle peut être obtenue en appliquant le prédicat;
- les contraintes de contenance (ayant comme hypothèse les mêmes conditions que l'identité) sont éliminées en enlevant d'une des deux relations les attributs en double. Les autres sont transformées en contrainte d'inclusion (on vérifie bien les conditions: deux tables différentes dans la même vue puisqu'il ne reste plus qu'une vue globale);
- les contraintes de disjonction sont devenues des contraintes d'exclusions.

  Les contraintes d'intégrations ne peuvent être enlevées que si elles sont redondantes: si ce n'est pas le cas, une procédure d'exception entièrement traitée par l'utilisateur est lancée. Celui-ci peut décider de faire les modifications adéquates ou d'arrêter l'intégration.

2.7 La démarche de *[Motro 87]*

L'auteur développe une technique d'intégration qui permet d'accéder à des schémas de différentes bases de données sans devoir à chaque requête reconstruire le chemin d'accès aux diverses composantes de la réponse.

Pour cela il mémorise les chemins déjà construits dans des vues virtuelles qu'il appelle **"superview"**.

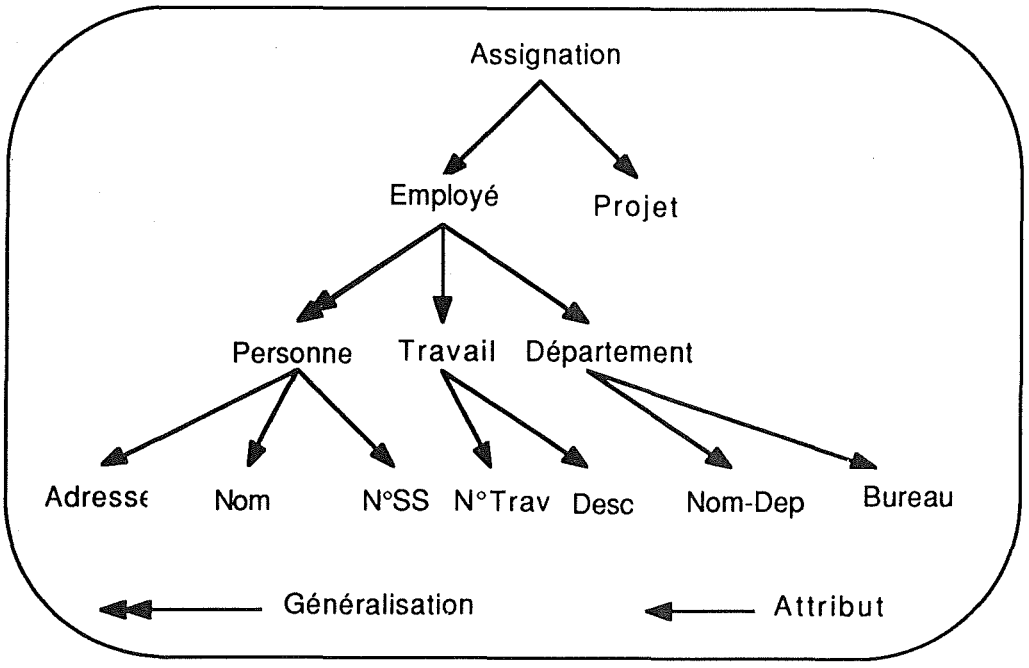
2.7.1 Le modèle de données

L'auteur choisit comme support de sa méthodologie un modèle fonctionnel, dont voici les principales définitions.
Une base de données est une collection D de classes possédant un nom et vérifiant les propriétés suivantes:

- deux relations sont définies sur D: ATT qui établit qu'une classe est attribut d'une autre et GEN qui établit qu'une classe est généralisation d'une autre. Ces deux relations sont mutuellement exclusives,
- chaque classe S a un domaine dom(S) de valeurs,
- chaque classe S a un type qui est l'ensemble des classes attributs associées à S,
- chaque classe S a une clé qui est sous-ensemble de son type,
- la relation T ATT S est supportée par une fonction de dom(S) vers dom(T),
- la relation T GEN S est supportée par une injection de dom(S) vers dom(T),
- l'héritage des attributs à travers la généralisation et la transitivité des généralisations se font par composition de fonctions,
- le produit cartésien des fonctions qui supportent les relations ATT réduit à la clé est lui une injection, et
- la notion de clé est héritée à travers des généralisations.

La Figure 2–9 illustre un exemple de représentation dans le modèle fonctionnel de *[Motro 87]*.

Figure 2–9: Exemple du modèle fonctionnel de *[Motro 87]*



2.7.2 La démarche

C'est une méthode binaire: on intègre les vues deux par deux et on mémorise le résultat dans des tables virtuelles[6].

On dispose de deux outils: un générateur de bases de données virtuelles et un processeur de requêtes virtuelles.

C'est l'utilisateur qui doit identifier les similitudes des deux schémas à intégrer: pour cela il doit connaître la signification des classes. Une fois qu'il a repéré les similitudes, il doit "restructurer" le schéma: en fait, il applique les différentes fonctions qui sont à sa disposition pour créer une table virtuelle (y compris les mappings) grâce au générateur de bases de données(BD) virtuelles.

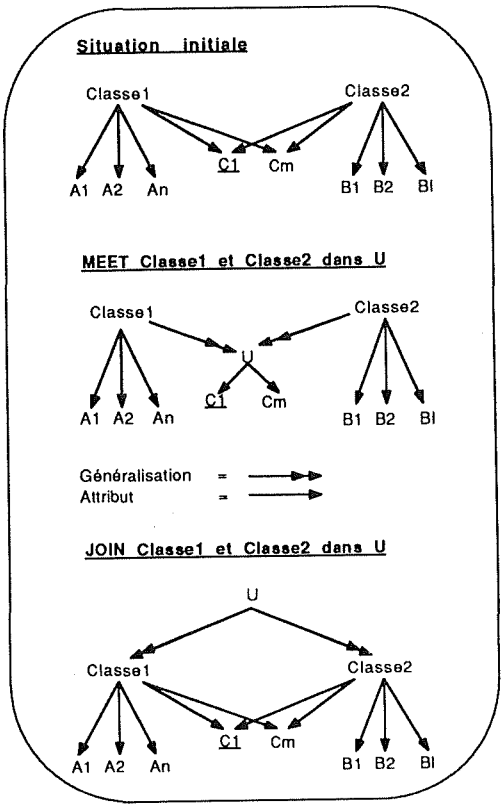
Une fois que les tables virtuelles (superviews) dont il a besoin sont générées (et uniquement celles-là, donc on n'a fait qu'une intégration partielle), on peut soumettre les requêtes au processeur qui va combiner les réponses pour essayer d'être le plus général possible. Il peut arriver, comme on peut travailler sur des BD existantes, que certaines réponses soient incompatibles et dès lors le processeur ne peut répondre à la question et renvoie la valeur **[not-consistent]**.

Les fonctions disponnibles

- L'opérateur MEET s'applique à deux classes ayant une clé commune et non reliées par la relation GEN. Dans ce cas, il produit une nouvelle classe qui est la généralisation des deux autres, qui a pour type l'intersection des types des deux autres et pour domaine l'union des domaines des deux classes auxquelles il s'applique.
  A la Figure 2–10, on peut voir l'effet de l'opérateur MEET appliqué à la situation initiale;
- l'opérateur JOIN s'applique dans les mêmes conditions. Il produit une nouvelle classe qui est sous-type des deux autres, qui a pour type l'union des types des deux autres et pour domaine l'intersection des domaines des deux classes auxquelles il s'applique.
  L'exemple de la Figure 2–10 illustre l'effet de l'opérateur JOIN appliqué à la situation initiale;
- l'opérateur FOLD s'applique sur une classe et la remplace par une autre.
  L'exemple Figure 2–11 montre l'effet de l'opérateur FOLD appliqué à la situation initiale;
- l'opérateur RENAME permet de changer le nom d'une classe (quand deux vues à intégrer ont une classe de même nom);
- l'opérateur AGGREGATE regroupe des classes attributs (de Classe2 par exemple) en une seule classe (qui reste attribut de Classe2).
  La Figure 2–11 montre l'effet de l'opérateur AGGREGATE sur le résultat de l'opérateur FOLD;
- l'opérateur TELESCOPE désagrège une classe attribut en ses attributs (inverse de AGGREGATE).
  La Figure 2–11 illustre l'effet de l'opérateur TELESCOPE sur le résultat de l'opérateur AGGREGATE;

[6] Elles ne contiennent pas de données mais des mappings pour accéder aux données des schémas d'origine.

Figure 2–10: Les opérateurs MEET et JOIN



- l'opérateur ADD ajoute une classe;
- l'opérateur DELETE détruit la relation ATT à laquelle participe une classe qui ne joue pas le rôle de clé. Si cette classe devient libre, elle est détruite aussi et on examine tous les attributs qu'elle pouvait posséder (pour voir s'il ne faut pas eux aussi les détruire).

Le mapping

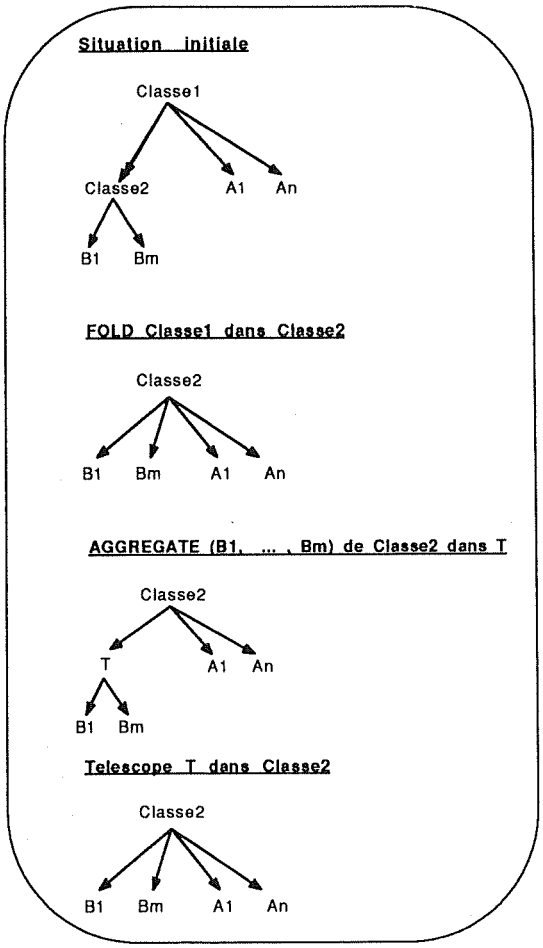
Il est exécuté au moment de l'application de l'opérateur introduit par l'utilisateur. Il associe à la classe produite une expression qui est une combinaison des expressions associées aux classes auxquelles l'opérateur s'applique. Par exemple "MEET S and T into Q" va associer à Q l'expression "expression de S ET expression de T".

La traduction des requêtes

Il existe dans le modèle de *[Motro 87]*, trois opérateurs d'accès aux données qui sont les mécanismes qui réalisent les requêtes:

- domain: {S} = dom(S) (il renvoie les valeurs du domaine de S),
- function: T(S=s) = Fst(s) (il renvoie, si T ATT S, la valeur t de dom(T) qui est associée à la valeur s de dom(S) comme attribut T par la fonction Fst qui supporte la relation ATT), et

Figure 2–11: Les opérateurs FOLD, TELESCOPE et AGGREGATE



- inverse: {S(T=t)} = l'inverse de Fst appliquée à t (il renvoie toutes les valeurs s1, ... sn de dom(S) qui ont t de dom(T) pour attribut).

L'opérateur function T(S=s) renvoie

— **[not-applicable]** si T n'est pas attribut de S,

— **[not-found]** si s n'appartient pas à dom (S), ou

— **[not-available]** si Fst(s) n'est pas défini,

— Fst(s) sinon.

L'opérateur inverse {S(T=t)} renvoie

— **[not-applicable]** si T n'est pas attribut de S,

— **[not-found]** si t n'appartient pas à dom(T), ou

— l'inverse de Fst appliqué à t sinon.

Grâce à ces opérateurs, le processeur de requêtes est capable de combiner les réponses de plusieurs demandes en fournissant toujours la solution la plus favorable. Comme les requêtes peuvent être imbriquées (car une superview, qui est déjà une requête complexe, peut faire partie d'une autre requête), le processus de construction de réponse doit être récursif. Il est constitué d'une routine de traduction pour chaque fonction de transformation et d'un pilote qui détecte le genre d'opérateur à traduire et envoie l'expression à la bonne routine de traduction.

2.8 La démarche de *[Larson & Al 89]*

Les auteurs proposent la conception d'un outil interactif qui effectue l'intégration en deux temps:

1. l'intégration des classes d'objets (type d'entités et catégorie),
2. l'intégration des types d'associations.

La solution proposée au problème de l'intégration ne permet pas de détecter les conflits entre les vues et ne considère pas les problèmes de terminologie[7].

La démarche choisie est celle de l'intégration des vues sur base de l'équivalence des attributs des objets de deux ou plusieurs schémas à intégrer.

2.8.1 Le modèle de données

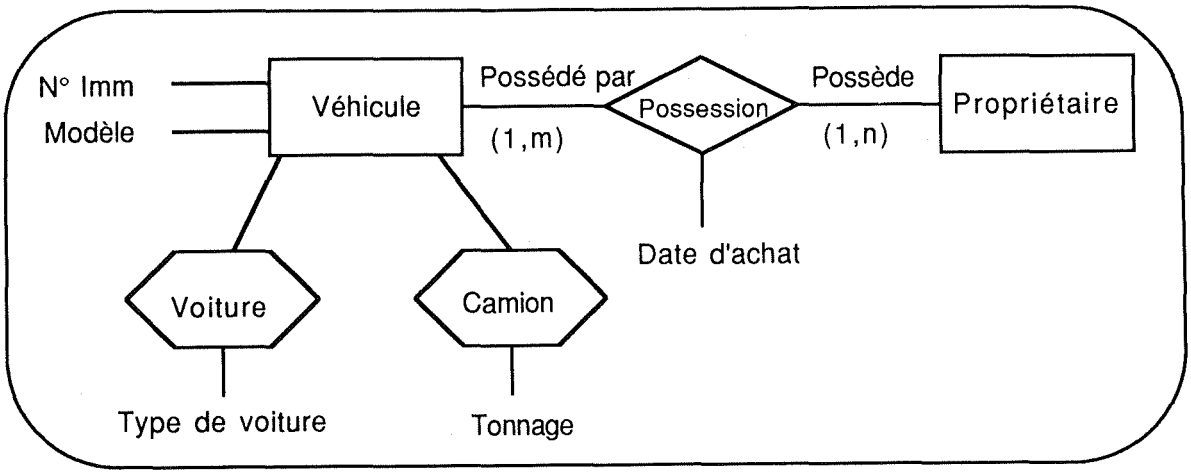
Le modèle de données utilisé est un modèle Entité/Association étendu appelé modèle ECR (Entity-Catégory-Relationship) dont les extensions par rapport au modèle de *[Chen 76]* sont:

- le concept de catégorie pour représenter les sous-classes (pour le modèle de données ECR, une catégorie peut aussi être utilisée pour représenter un sous-ensemble de l'union des classes d'objets),
- les contraintes de cardinalité et de dépendance sur les types d'associations.

Le modèle ECR utilise les concepts de type d'entités, de type d'associations, de catégorie et d'attribut. Le terme "classe d'objets" représente soit un type d'entités, soit une catégorie. Un type d'entités est un ensemble d'entités qui ont les mêmes attributs. Une catégorie peut être utilisée pour modéliser une sous-classe d'une classe d'objets. Une catégorie hérite des attributs de la classe d'objets pour laquelle elle est une sous-classe.
L'exemple de la Figure 2–12 montre la représentation graphique du modèle ECR. Les rectangles représentent des types d'entités, les hexagones aplatis représentent des catégories, les losanges représentent les types d'associations.

[7] Ce problème consiste à détecter les conflits entre les noms des objets.

Figure 2–12: Exemple de schéma ECR



2.8.2 La démarche

La démarche pour l'intégration des vues utilisant le modèle de données ECR est d'intégrer en premier lieu les classes d'objets (types d'entités et catégories) et ensuite d'intégrer les types d'associations. Dans cette démarche on remarquera l'importance du rôle joué par la spécification de l'équivalence entre les attributs, dans les deux étapes. Dans la première étape, l'intégration de classes d'objets est accomplie par la détermination de l'équivalence entre les identifiants de ces classes d'objets, ensuite suivie par l'intégration des types d'associations.

Spécification de l'équivalence sur les attributs

Cette étape est réalisée manuellement par l'administrateur de base de données. Celui-ci établit les équivalences entre les attributs à l'aide de critères de comparaison. Ces critères sont l'ensemble de toutes les contraintes définies par le concepteur sur les attributs, à savoir: l'unicité de l'attribut, ses cardinalités, son domaine, les contraintes d'intégrité statiques et dynamiques dans lesquelles il est impliqué, les contraintes de sécurité qui définissent les droits d'accès à cet attribut et les opérations autorisées sur cet attribut. Deux attributs sont équivalents s'il existe un isomorphisme entre deux sous-ensembles non vides de valeurs des deux attributs, c'est-à-dire une bijection qui conserve toutes les contraintes définies.

Il existe trois cas possibles d'équivalence d'attributs:

- **égalité** des domaines $DOM(a) = DOM(b)$ où a et b sont des attributs appartenant à des schémas différents,
- **inclusion** d'un domaine dans l'autre $DOM(a) \supset DOM(b)$ ou l'inverse,
- **recouvrement** des domaines $DOM(a) \cap DOM(b) \neq \{\}$ mais aucun des deux domaines n'est inclus dans l'autre.

Détection des équivalences entre classes d'objets

L'approche choisie pour la détermination d'équivalences entre les classes d'objets se base sur les spécifications de l'équivalence des attributs. Les classes d'objets auront la même équivalence que leurs attributs identifiants.

Les différents cas d'équivalence entre deux classes d'objets A et B, où A et B sont identifiés respectivement par k_1 et k_2 et ont respectivement comme attributs $a_1, ..., a_m$ et $b_1, ..., b_n$ sont:

- si $DOM(k_1) = DOM(k_2)$, dans ce cas les classes d'objets A et B peuvent être intégrées dans une simple classe avec comme identifiant unique k_1 ou k_2,
- si $DOM(k_1) \supset DOM(k_2)$ ou l'inverse, alors l'un des deux objets devient sous-classe de l'autre, ou
- si $DOM(k_1) \cap DOM(k_2) \neq \{\}$, dans ce cas on crée un nouvel objet "sur-type" de A et B dont A et B deviennent des catégories.

Cette détermination des équivalences d'objets impose des restrictions pour les équivalences d'attributs. Soit "a" un attribut de la classe d'objets A et "b" un attribut de la classe d'objets B alors les seules équivalences possibles sont:

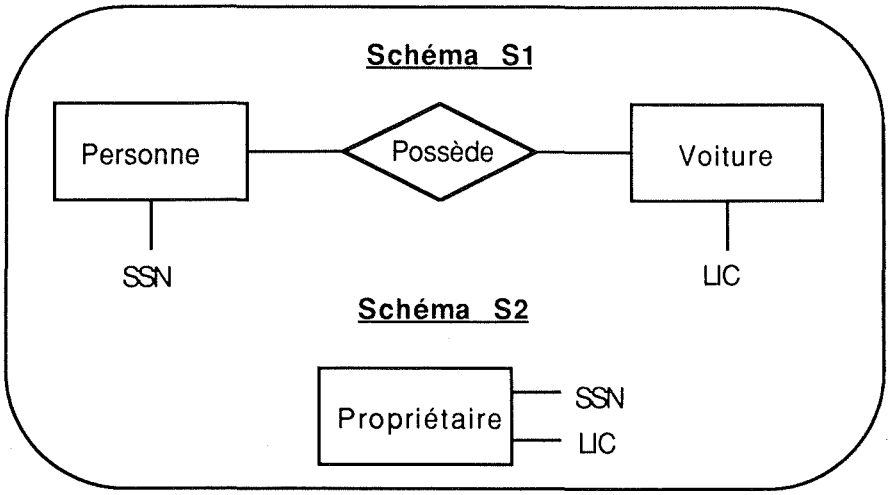
- s'il y a une équivalence d'égalité entre les objets A et B alors la seule équivalence d'attributs entre a et b est une équivalence d'égalité, ou bien
- s'il y a une équivalence d'inclusion entre les objets A et B ou l'inverse, alors soit:
  - — il existe une équivalence d'inclusion entre les attributs a et b, ou
  - — il existe une équivalence d'égalité entre les deux attributs.

Détermination de l'équivalence entre types d'associations

L'équivalence entre types d'associations peut être définie de la même manière que l'équivalence entre classes d'objets. La différence se situe au niveau de la détermination de l'équivalence entre types d'associations, l'analyse ne porte plus sur l'équivalence des identifiants des objets à intégrer, mais porte sur l'agrégation des identifiants des objets participants aux types d'associations.

Normalement, l'intégration des types d'associations porte sur des types d'associations de même degré (=2, car souvent la politique choisie est de mettre les schémas sous la forme canonique avant la phase d'intégration). Cependant, il est possible d'intégrer des types d'associations de degrés différents. Dans ce cas, les auteurs proposent de toujours faire apparaître dans le schéma intégré le type d'associations dont le degré est le plus élevé.
Le type d'associations (voir Figure 2–13) "Possède" est de degré 2; l'autre type d'associations "Propriétaire" est de degré 1 (car le type d'associations est lui-même un type d'entités). A cause de la bijection des valeurs des attributs identifiants (SSN et LIC) de propriétaire et l'agrégation des identifiants (SSN et LIC) des classes d'objets (Personne et Voiture), le type d'associations possède peut être intégré avec le type d'entités propriétaire. Le schéma intégré reprendra le type d'associations dont le degré est le plus haut, c'est-à-dire le type d'associations "possède".

Figure 2–13: Integration de types d'associations de degrés différents



Stratégies d'intégration

Une fois que les équivalences sur les objets et les types d'associations sont établies, et cela à l'aide des spécifications des équivalences d'attributs, nous pouvons intégrer les classes d'objets des schémas de quatre façons différentes. En ce qui concerne les types d'associations, il existe une approche un peu différente de celle des classes d'objets.

Stratégie 1 Cette stratégie consiste à intégrer tous les attributs non disjoints: quand deux classes d'objets (soient A et B) sont intégrées, les attributs équivalents (soient a et b) sont toujours intégrés sur une nouvelle forme d'attribut c avec comme domaine DOM(c) = DOM(a) ou DOM(c) = DOM(b). La nouvelle classe d'objets créée (soit AB') représente l'intégration des deux classes d'objets A et B. La classe d'objets AB' comprend l'union des univers du discours de A et B.

Le but essentiel de cette stratégie est d'obtenir un schéma intégré avec le minimum d'attributs. Les utilisateurs qui choisissent cette stratégie veulent éviter la complexité d'avoir dans le schéma intégré des attributs similaires.

Stratégie 2 On intègre seulement les attributs qui ont une équivalence d'égalité entre eux. Cette autre approche d'intégration, consiste lors de l'intégration à garder le maximum d'attributs dans le schéma intégré.

Cette stratégie fait la distinction entre deux cas:

— soit il y a une équivalence d'égalité entre les deux attributs à intégrer. L'effet est le même que celui de la première stratégie;

— sinon les deux attributs sont repris indépendamment l'un et l'autre dans le schéma intégré.

Cette stratégie peut rendre le schéma un peu plus complexe par son grand nombre d'attributs dans le schéma intégré. L'inconvénient est que l'équivalence explicite entre deux attributs similaires n'est pas présente dans le schéma intégré. Cela rend plus difficile la formulation de certaines requêtes à cause de l'absence complète de description des équivalences d'attributs dans le schéma final.

Stratégie 3 Cette stratégie élimine l'inconvénient de la deuxième stratégie par la description des équivalences d'attributs non reprises dans le schéma final. Cette stratégie peut avoir un peu plus d'attributs dans le schéma intégré, en comparaison avec la deuxième stratégie. Ces attributs supplémentaires et la description explicite de l'équivalences des attributs donnent à l'utilisateur plus d'information sémantique entre les attributs similaires non intégrés dans le schéma final.

Stratégie 4 On intègre tous les attributs non disjoints et on accomplit des migrations de valeurs entre les attributs. La stratégie 4 produit le même résultat que la stratégie 1, mais à ceci près qu'elle permet que les valeurs communes des deux attributs à intégrer n'apparaissent qu'une seule fois dans le schéma intégré. Cette stratégie applique le mécanisme de généralisation sur les valeurs des attributs.

Intégration des types d'associations On intègre le type d'associations comme s'il représentait un attribut (monovalué ou multivalué) pour chaque classe d'objets qui participe dans le type d'associations. Cet attribut est aussi un attribut agrégé puisqu'il comprend les attributs du type d'associations et les attributs identifiants des classes d'objets qui participent à l'association.

On peut ainsi intégrer les types d'associations en les traitant comme des attributs agrégés et on utilise la stratégie 3 ou la stratégie 4. On dit que deux attributs agrégés ont une équivalence d'égalité entre eux s'il existe une équivalence d'égalité entre chaque attribut participant à l'agrégation de l'attribut agrégé. Après l'intégration des attributs agrégés, on les remplace par le type d'associations correspondant.

2.9 La démarche de *[Bouzeghoub & Al 90]*

La méthode d'intégration des vues définie par les auteurs est basée sur un mécanisme d'unification de la sémantique des schémas. Ce mécanisme est inspiré de l'unification logique utilisée en PROLOG. Mais les deux mécanismes se différencient par plusieurs aspects:

- les schémas n'ont pas de variable, alors l'unification structurelle ne procède pas par substitution des variables comme cela est fait dans l'unification logique mais bien par analogie des structures;
- l'unification structurelle n'a pas pour seul but de trouver la stricte égalité entre deux vues, mais comprend aussi la partie des détections de contradictions, de recouvrements et de disjonctions;
- l'unification structurelle requiert, chaque fois que cela est nécessaire une intervention du concepteur.

L'objectif de cette démarche d'intégration de vues est de

— détecter les conflits possibles,

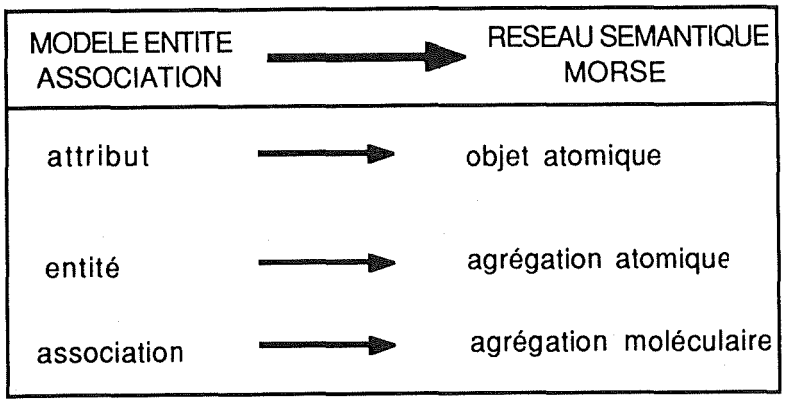
— éviter les redondances et les conflits de structures entre les différentes vues, et

— obtenir un schéma global unique représentant l'union des univers du discours des différentes vues intégrées.

2.9.1 Le modèle de données

Le modèle utilisé est le modèle Entité/Association de *[Chen 76]* étendu. Ce modèle contient les concepts usuels d'attribut, d'entité, d'association et de hiérarchie de généralisation avec les règles d'héritage qui les caractérisent. La formalisation interne de ce modèle est faite sous forme d'un réseau sémantique (appelé MORSE *[Bouzeghoub 84]*). La Figure 2–14 montre l'équivalence entre certains concepts des deux modèles. Le réseau sémantique MORSE permet la représentation de tous les concepts du modèle Entité/Association. Il est sémantiquement plus riche, dans la mesure où il permet en plus la représentation des notions de classe d'objets, de hiérarchie de classes d'objets ainsi que les cardinalités des attributs et des associations.

Figure 2–14: Représentation en MORSE des concepts du modèle Entité/Association



Le réseau sémantique MORSE est défini par le triplet **RS(N,A,C)** où **N** est un ensemble de symboles désignant les noeuds du réseau, **A** est un ensemble d'arcs représentant des relations binaires définies entre les noeuds du réseau, et **C** est un ensemble de contraintes associées aux noeuds ou aux arcs.

L'ensemble **N** est constitué d'objets décrivant les attributs, les entités, et associations des schémas.

L'ensemble d'arcs **A** est constitué des types de relations suivants:

— p(X,Y) qui signifie que Y est un constituant atomique de X,

— o(X,Y) qui signifie que Y est un constituant moléculaire de X,

— c(x,X) qui signifie que x est un élément de la classe X,

— g(X,Y) qui signifie que X est une sous-classe de la classe Y.

On peut citer parmi les contraintes de l'ensemble **C**, les contraintes de:

— domaine qui caractérisent les attributs (ou classes d'objets atomiques). Les domaines de base sont: entiers, réels, booléens, chaînes de caractères ou en extension par énumération de ses valeurs;

— cardinalité qui portent sur les relations p et o: c'est un couple de valeurs [m,n] qui spécifie si la relation est totale ($m \geq 1$) ou partielle ($m=0$) et si elle représente une fonction monovaluée ($n=1$) ou multivaluée ($n \geq 1$);

— dépendance fonctionnelle qui relient des couples d'attributs ou des collections d'attributs;

— intersection ou disjonction de classes: deux classes distinctes X et Y sont dites disjointes si et seulement si $\forall x : c(x, X) \Longrightarrow \neg c(x, Y)$, autrement elles sont dites non disjointes.

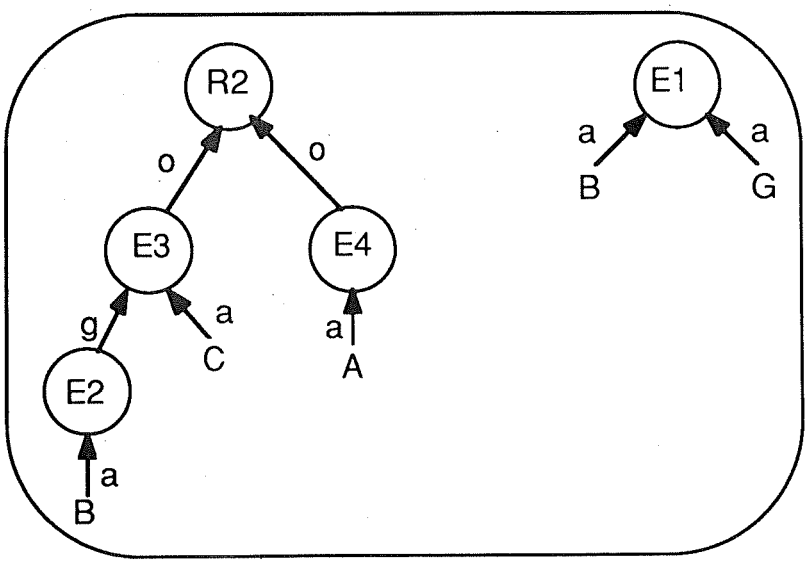
Pour exemple, soit la vue avec les objets suivants:

l'entité **E1** avec deux attributs: B et G,
l'entité **E2** avec un attribut: B,
l'entité **E3** avec un attribut: C,
l'entité **E4** avec un attribut: A,
l'entité **E2** est une sous-classe de l'entité **E4**, et
l'association **R2** entre l'entité **E3** et l'entité **E4**.

La représentation graphique de ce réseau sémantique est faite à la Figure 2–15. Tandis que la représentation de cet exemple en MORSE est constituée d'un ensemble d'arcs:

$$a(E1, B) \wedge a(E1, G) \wedge a(E2, B) \wedge a(E3, C) \wedge a(E4, A) \wedge o(R2, E3) \wedge o(R2, E4) \wedge g(E4, E2).$$

Figure 2–15: Représentation graphique d'un réseau sémantique



2.9.2 La démarche

L'intégration de deux vues, en l'occurrence de deux réseaux sémantiques, vise à faire coïncider les deux parties identiques (schématiquement à les superposer). Mais les problèmes de terminologie, de définition de classes et de contraintes font que ce processus peut rarement se réduire à une simple superposition de deux parties des schémas. L'intersection des deux réseaux doit être détectée et le choix de la représentation unique de cette intersection doit être fait. Le processus d'unification proposé par les auteurs s'apparente à un processus d'unification de deux parties de réseaux sémantiques.

Les auteurs ont élaboré un algorithme d'unification de vues qui a pour but de mettre en correspondance les éléments des deux vues qui semblent représenter le même concept.

Pour construire leur démarche d'unification sémantique, les auteurs ont défini une procédure d'identification syntaxique des termes, et une procédure de raisonnement par analogie qui conduit à caractériser deux objets par un vecteur de similitude dont les composantes sont des coefficients de similitude entre les noms d'objets, leurs structures respectives, leurs contraintes et leurs populations.

Vecteur de similitude

L'unification est basée sur trois notions: l'équivalence, la similitude, la dissemblance.

1. **équivalence** sémantique de deux concepts: cette notion dépend de la nature des deux concepts à comparer. On peut définir les équivalences sémantiques suivantes:
   * deux attributs sont équivalents s'ils ont même nom et même domaine,
   * deux entités sont équivalentes si elles sont constituées d'attributs équivalents, et
   * deux associations sont équivalentes si elles sont constituées d'entités (ou d'associations) équivalentes et d'attributs équivalents.
2. **similitude** sémantique entre deux concepts: deux concepts (attribut, entité ou association) sont similaires s'ils ont même nom, ou même domaine, ou même structure, ou s'ils font référence aux individus de la même population (univers du discours). La similitude n'est pas représentée par une caractéristique unique mais par un ensemble de caractéristiques qui peuvent définir un degré de similitude plus ou moins élevé selon le nombre et la nature de ces caractéristiques.
3. **dissemblance** sémantique entre deux concepts: deux concepts sont dissemblables s'ils ne sont ni équivalents, ni similaires. La dissemblance entre deux concepts correspond à une similitude de degré le plus faible.

Raisonnement par analogie

Les vecteurs de similitude sont le résultat d'un raisonnement par analogie fait par l'unificateur de structures. Le principe de ce raisonnement par analogie est basé sur quatre mécanismes fondamentaux:

1. une **comparaison** terme à terme des constituants des deux concepts à unifier: cette comparaison se fait d'abord sur les noms respectifs des deux concepts à unifier, puis sur leurs domaines s'il s'agit d'attributs ou leurs structures s'il s'agit d'entités ou d'associations. Enfin, on peut compléter par les contraintes de dépendance et de cardinalité.
2. une **déduction** par héritage pour renforcer le degré de similitude entre deux concepts: les structures de deux concepts à unifier ne sont pas équivalentes mais présentent des similitudes, on peut tenter de les rendre équivalentes ou renforcer leur vecteur de similitude en leur appliquant à l'un ou à l'autre ou aux deux concepts les règles d'héritages s'ils sont impliqués dans des hiérarchies de généralisation. Par exemple, Vue1: $a(M1, A) \wedge a(M1, B) \wedge a(M1, C)$ et Vue2: $a(M1, A) \wedge a(M2, B) \wedge a(M2, C)$ comme M1 est une sous-classe de M2 dans la Vue2, après héritage de B et C elle devient équivalente de M1 dans Vue1.

3. une **transformation** de structures qui permet de passer d'une représentation à une autre pour rechercher l'équivalence ou une similitude plus grande entre les deux concepts. Ceci peut être le cas des associations qui peuvent apparaître sous différentes formes dans différentes vues. Par exemple, dans la vue V1 on peut avoir l'association ternaire R(E1,E2,E3), et dans la vue V2 on peut avoir les trois associations binaires R1(E1,E2), R2(E1,E3) et R3(E2,E3). Par une transformation, on peut transformer l'association R en trois associations binaires R1, R2, R3. Après cette transformation, on peut revenir à la comparaison terme à terme des différentes structures.
4. une **interaction** avec le concepteur si les actions précédentes ne sont pas suffisantes pour évaluer les composantes du vecteur de similitude. Cette interaction se fait uniquement si les deux techniques précédentes ne contribuent plus à renforcer le vecteur de similitude. Cette interaction se fait pour assigner une valeur à la composante "instances" qui renseigne sur la population représentée.

Stratégie de sélection des objets à unifier

Le processus d'unification commence avec les objets les plus structurés (e.g. d'abord les associations, suivi des entités et enfin les attributs).

On commence par comparer les associations de même nom, puis les autres associations en partant du niveau le plus haut dans la hiérarchie puis les entités ne participant à aucune association (entités libres). Considérer d'abord les objets de même nom consiste à admettre l'heuristique selon laquelle les conflits de terminologie (synonyme et homonyme) ne sont pas systématiques. D'autre part, on considère d'abord les associations de niveau le plus haut dans la hiérarchie puisque le processus d'intégration choisi par les auteurs consiste à "intégrer par niveaux": pour intégrer les objets les plus composés, on intègre leurs composants: le principe est donc respecté pour les éventuelles associations d'associations. Cette stratégie est résumée par l'algorithme de la Figure 2–16.

2.10 La démarche de *[Spaccapietra & Parent 90]*

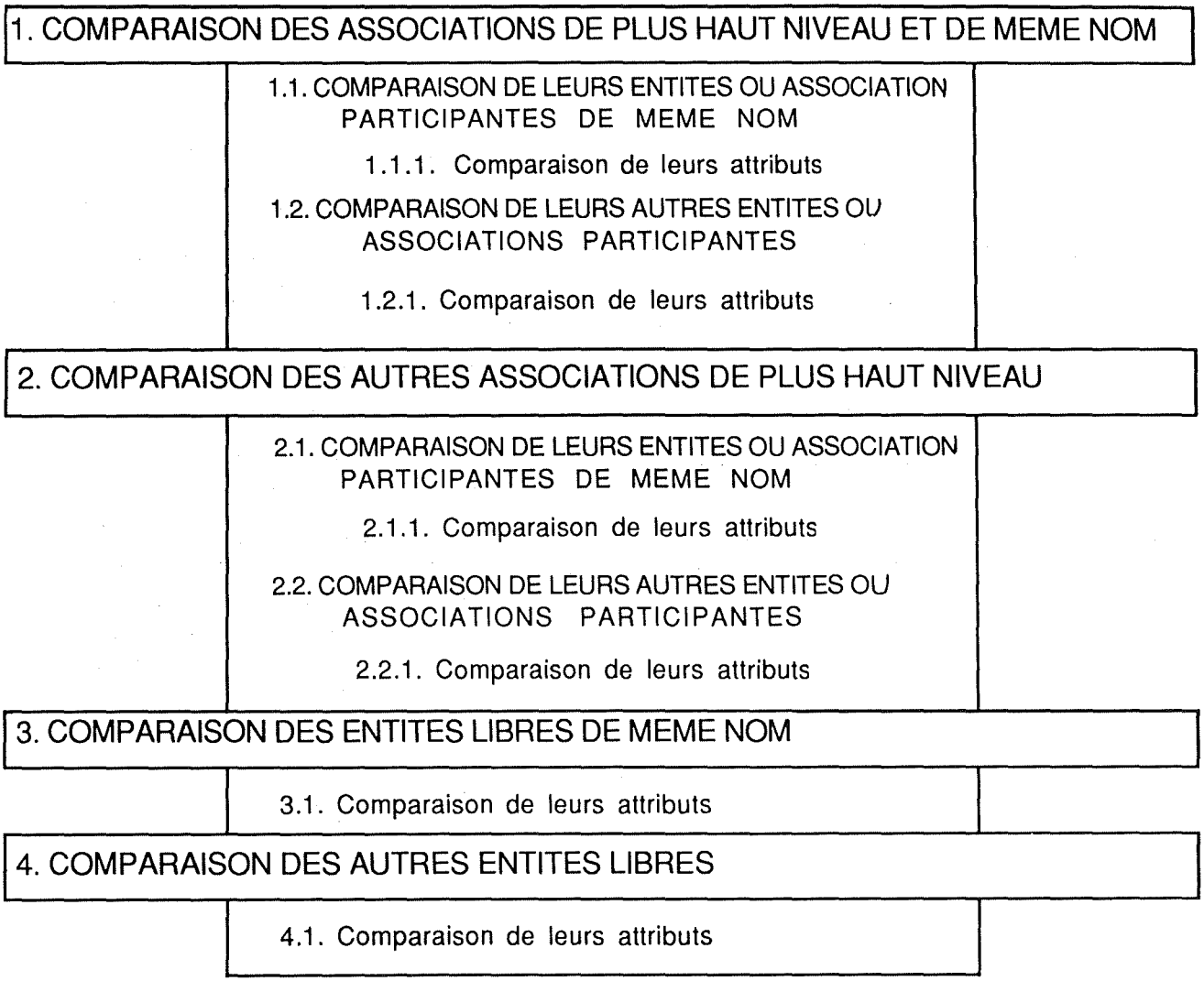
Les auteurs se concentrent uniquement sur la résolution des conflits pour fournir aux utilisateurs confrontés aux problèmes de l'intégration une solution en profondeur et complète.

Ils essaient cependant d'être assez formels et généraux afin que leur méthode n'ait pas une application limitée. Les caractéristiques de cette méthode sont:

- la résolution de manière automatique des conflits structurels,
- la résolution des conflits sans modification des vues initiales,
- l'approche déclarative formelle pour permettre à un utilisateur (ou un administrateur de bases de données) de déclarer les correspondances entre schémas,
- applicable à une grande variété de modèles, et
- la génération automatique des mappings structurels et opérationnels entre les vues et le schéma intégré.

Une démarche d'intégration n'est pas entièrement automatisée car il faut savoir ce qui est à intégrer. Une fois cela déterminé (par l'utilisateur), un outil peut intégrer sans plus se poser de questions les vues qu'on lui a données en entrée.

Figure 2–16: Algorithme d'intégration des objets des vues



Le fait de préserver les vues peut s'avérer capital: si des programmes ont déjà été conçus sur un schéma partiel, ils peuvent être réutilisés sur le schéma intégré.

Comme la méthodologie est décrite dans un modèle sémantique et en termes généraux, elle est indépendante des modèles utilisés pour définir les vues à intégrer. Des recherches se poursuivent pour adapter la méthodologie dans un cadre orienté objet.

La génération automatique se fait par un algorithme qui exécute l'intégration dans tous les cas concevables. Cependant, comme la recherche est récente et pas encore arrivée à son terme, l'algorithme n'est pas encore implémenté quoique les principes, pour les objets équivalents, en sont déjà bien définis (en termes conceptuels).

2.10.1 Le modèle de données

Le modèle utilisé est une extension du modèle Entité/Association *[Chen 76]* que Spaccapietra appelle ***Entity Relationship for Complex objects*** (ERC +). Il contient les éléments suivants:

- des entités formées d'un ensemble d'attributs,
- des associations (éventuellement cycliques) entre des entités avec des attributs éventuels,
- des noms de rôles pour chaque participation d'une entité à une association. Ces rôles ont une connectivité minimale et maximale,
- les attributs peuvent être obligatoires ou optionnels, monovalués ou multivalués, atomiques ou complexes,
- les entités et les associations peuvent avoir un ou plusieurs ensembles d'attributs comme identifiants, et
- le modèle supporte deux généralisations: l'association "is-a" qui correspond au concept de généralisation et l'association "may-be-a" qui a la même sémantique mais qui ne requiert pas de dépendance d'inclusion entre le type et le sur-type et ne fournit pas automatiquement le mécanisme d'héritage.

2.10.2 La démarche

C'est une démarche binaire: on sélectionne deux vues à intégrer, on établit les correspondances puis on réalise l'intégration.

C'est l'utilisateur qui introduit le mapping structurel entre les objets des différentes vues car il est le seul à connaître la sémantique de chaque objet. Comme la même sémantique peut être exprimée avec des noms différents (conflit de terminologie) ou des modélisations différentes (conflit de structures), la démarche (qui veut être générale) doit être définie en termes plus abstraits. Elle permet la comparaison d'objets à priori différents.

Une fois les correspondances définies entre les objets, un algorithme peut prendre en charge automatiquement l'intégration. Cela suppose que l'utilisateur, qui a introduit ces correspondances, a été complet et qu'il a levé toute ambiguïté.

Les correspondances existantes

Définitions sous-jacentes

1. L'univers du discours d'un type d'entités E:
   c'est l'ensemble des objets du monde réel que les occurrences de E représentent. Il y a une correspondance biunivoque entre le type d'entités et son univers du discours;
2. l'univers du discours d'un type d'associations R liant E1, . . . , En:
   c'est le super ensemble (car un même élément peut être présent plusieurs fois) de n-uplets [O1, . . . , On] d'objets du monde réel où chaque Oi est une occurrence du type d'entités Ei et où les Oi d'un même n-uplet sont reliés par une association, représentée par R, dans l'univers du discours. Il y a une correspondance biunivoque entre le type d'associations et son univers du discours;

3. l'univers du discours d'un attribut A:
   c'est l'ensemble des objets du monde réel que les valeurs de A représentent. Il y a une fonction surjective totale entre A et son univers du discours car par exemple le concept de mois peut être représenté par différentes valeurs (entiers de 1 à 12, chaînes de caractères, . . .);
4. notion de lien
   Si X et Y sont deux éléments liés quelconques (attribut, type d'entités ou type d'associations) d'un schéma et que l'un des deux est un attribut, alors X-Y est un lien attribut autrement c'est un lien rôle;
5. notion de chemin
   Un chemin X1- . . . -Xn est une succession de liens Xi-Xi+1 entre des éléments quelconques d'un schéma;
6. l'univers du discours d'un chemin X1- . . . -Xn
   C'est le super ensemble de paire [O1,On] d'objets du monde réel où O1 et On appartiennent respectivement à l'univers du discours de X1 et Xn. Il faut aussi qu'il existe des objets Oi des univers de discours intermédiaires qui soit reliés entre eux par des associations du monde réel du même type que Xi-Xi+1.

Assertions de correspondance

Sur base de ces définitions, on peut établir des correspondances:

1. assertion d'équivalence entre deux éléments X1 et X2
   Il faut que leurs univers du discours respectifs soient égaux (s'ils sont de même type) ou liés par une fonction bijective totale entre les deux univers du discours qui associe les éléments avec la même sémantique;
2. assertion d'inclusion entre deux éléments quelconques
   Si on veut dire que X1 contient X2, il faut que l'univers du discours (UDD) de X1 contienne celui de X2 (s'ils sont de même type) ou qu'il existe une fonction injective totale de UDD(X2) vers UDD(X1) qui associe les éléments avec la même sémantique;
3. assertion d'intersection entre deux éléments X1 et X2
   Il faut que leurs UDD respectifs ne soient pas disjoints (s'ils sont de même type) ou qu'il existe une fonction injective partielle entre les deux UDD qui associe les éléments avec la même sémantique;
4. assertion d'exclusion entre deux éléments quelconques
   Il faut que leurs UDD respectifs soient disjoints (s'ils sont de même type) ou qu'il n'existe pas de fonction entre les deux UDD qui associe les éléments avec la même sémantique;
5. assertion d'attributs correspondants
   Pour chaque objet en correspondance, on peut ajouter la partie des attributs que chacun possède qui sont aussi en correspondance. Un objet attribut possède un seul attribut: lui-même. Les correspondances qu'on peut définir entre attributs sont l'égalité (éventuellement à une fonction près), l'inclusion, l'intersection ou l'exclusion;
6. assertion d'équivalence entre deux chemins
   Il faut que leurs univers du discours respectifs soient égaux (s'ils sont de même type) ou liés par une fonction bijective totale entre les deux univers du discours qui associe les chemins avec la même sémantique et qui assure en plus que l'origine et la destination de ces chemins soient les mêmes;

7. assertion d'inclusion entre deux chemins
   Si on veut dire que C1 contient C2, il faut que l'univers du discours de C1 contienne celui de C2 (s'ils sont de même type) ou qu'il existe une fonction injective totale de UDD(C2) vers UDD(C1) qui associe les chemins avec la même sémantique et qui assure en plus que l'origine et la destination de ces chemins soient les mêmes;
8. assertion d'intersection entre deux chemins
   Il faut que leurs UDD respectifs ne soient pas disjoints (s'ils sont de même type) ou qu'il existe une fonction injective partielle entre les deux UDD qui associe les chemins avec la même sémantique et qui assure en plus que l'origine et la destination de ces chemins soient les mêmes;
9. assertion d'exclusion entre deux chemins
   Il faut que leurs UDD respectifs soient disjoints (s'ils sont de même type) ou qu'il n'existe pas de fonction entre les deux UDD qui associe les chemins avec la même sémantique;

Règles d'intégration

Ces règles respectent les deux principes suivants: elles sont applicables quel que soit le modèle et en cas de conflit, la solution la moins restrictive est adoptée. Il faut noter que pour le moment, ces règles ne s'appliquent que pour des objets qui sont équivalents.

1. Intégration de deux éléments
   Si deux éléments sont équivalents, ils sont intégrés en un type d'entités s'ils sont de nature différente ou en un objet de leur type (si ces deux éléments ne sont pas des attributs)[8];
2. intégration de deux liens
   Si deux liens sont équivalents et si leurs origines et destinations sont aussi équivalentes, le résultat sera une origine et une destination intégrées comme dans le point ci-dessus et entre les deux,
   - un lien rôle si l'origine et la destination sont un type d'entités et un type d'associations,
   - un lien attribut si l'origine ou la destination est un attribut, ou
   - un type d'associations et ses deux rôles si l'origine et la destination sont deux types d'entités.

   La cardinalité du lien créé (par exemple A-B résultat de A1-B1 et A2-B2) sera

   cardmin(A) = min (cardmin(A1), cardmin(A2))
   cardmax(A) = max (cardmax(A1), cardmax(A2))
   cardmin(B) = min (cardmin(B1), cardmin(B2))
   cardmax(B) = max (cardmax(B1), cardmax(B2));
3. intégration de deux chemins
   Si l'un des deux est un lien, on intègre la source et la destination comme ci-dessus mais seul le chemin se retrouve dans le schéma intégré (éventuellement corrigé si les cardinalités du lien sont plus restrictives).
   Si les deux sont des chemins, on copie les deux chemins dans le schéma intégré mais il faut ajouter une contrainte disant que les deux chemins doivent aboutir à la même destination si on part de la même source;

[8] Pour les autres correspondances, l'auteur renvoie à une analyse plus détaillée comme celle trouvée dans *[Larson & Al 89]*.

4. intégration d'attributs d'éléments correspondants
   Pour les attributs communs des éléments (par exemple Ai1 et Ai2), ils sont recopiés à l'élément intégré avec comme cardinalité minimale le minimum de cardmin (Ai1) et de cardmin (Ai2). De même, la cardinalité maximale sera le maximum de cardmax (Ai1) et de cardmax (Ai2).
   Les attributs (Bj) n'appartenant qu'à un des deux éléments sont ajoutés à l'élément intégré avec comme cardinalité [0 : cardmax (Bj)];
5. intégration d'attributs en bout de chemins correspondants
   Si l'un des deux est un lien (attribut), seul le chemin se retrouve dans le schéma intégré. On ajoute à l'objet en fin de chemin l'attribut avec ses cardinalités minimale (ou 0 si l'objet est présent dans plusieurs vues) et maximale dans le schéma intégré.
   Si les deux sont des chemins, on ajoute aux objets en fin de chemin leur attribut respectif avec ses cardinalités minimale (ou 0) et maximale dans le schéma intégré mais il faut ajouter une contrainte disant que les deux chemins doivent aboutir à la même valeur d'attribut si on part de la même source;
6. ajout de règle
   Pour tout lien (rôle ou attribut) non encore traité, on ajoute (si ce n'est pas déjà fait) sa source et destination dans le schéma intégré et on ajoute le lien avec comme cardinalité les mêmes que dans le schéma d'origine (sauf dans le cas où la source/destination était déjà dans le schéma: dans ce cas, le min devient 0).

Algorithme d'intégration proposé

Il reçoit en entrée deux vues et les assertions les concernant et il produit un schéma intégré et les correspondances (mapping opérationnel) entre le schéma intégré et les deux vues d'origine.

1. Préparation
   Enlever de l'ensemble des correspondances celles concernant des attributs et enlever temporairement des vues les attributs impliqués
   Enlever aussi les correspondances de chemins se terminant par un attribut qui participe aussi à une assertion entre attributs,
2. intégration des éléments
   - Phase 1
     Pour chaque correspondance entre éléments, exécuter la règle 1 (intégration des éléments), la règle 4 (intégrations de leurs attributs), marquer comme traité ces éléments, leurs attributs et leurs liens. Enfin générer deux mappings (un pour chaque vue) entre l'élément intégré et l'élément provenant de la vue en entrée,
   - Phase 2
     Pour tous les autres éléments non traités, les ajouter dans le schéma intégré avec leurs attributs et liens (grâce à la règle 6), les marquer et générer le mapping entre l'élément intégré et la vue dont il provient;
3. intégration des chemins
   - Phase 1
     Pour chaque assertion d'équivalence entre deux types d'associations et que ces types d'associations lient des types d'entités équivalents, il faut rendre explicite (i.e. ajouter des assertions à l'ensemble des correspondances) l'équivalence de rôle entre ces types d'associations et ces types d'entités,

— Phase 2
Pour chaque correspondance entre chemin, appliquer la règle d'intégration 2 (pour chemin de longueur 1 i.e. lien) ou 3, marquer les chemins traités et générer le mapping entre le chemin intégré et le chemin d'origine dans la première vue (idem pour la vue deux),

— Phase 3
Pour tous les autres chemins non traités, les ajouter dans le schéma intégré (grâce à la règle 6), les marquer et générer le mapping entre l'élément intégré et la vue dont il provient,

4. intégration des attributs
Pour chaque assertion sur les attributs (celles retirées dans la préparation), s'il existe une assertion sur des chemins impliquant ces attributs, appliquer la règle 5. Autrement, ajouter ces attributs dans le schéma intégré. Dans les deux cas, générer le mapping,

5. optimisation
Pour chaque type d'entités dont tous les rôles sont de cardinalité 1:1, le remplacer par un type d'associations.

CHAPITRE 3

SYNTHESE DES PROBLEMES SUR L'INTEGRATION DES VUES

3.1 Introduction

Après la présentation des différentes approches du problème de l'intégration de schémas de bases de données, nous exposerons dans ce chapitre les difficultés qui peuvent surgir au cours de l'élaboration d'une démarche d'intégration. Nous y exposerons aussi des concepts (stratégies n-aire et binaire, . . .) qui nous serviront dans le choix et le développement d'une méthodologie pour un contexte d'atelier logiciel (Chapitre 5).

3.2 Les modèles conceptuels utilisés

Nous présentons dans cette section les différents modèles qu'on emploie le plus souvent pour représenter des bases de données et que l'on retrouve aussi dans les différentes démarches passées en revue dans le chapitre précédent.
Une caractéristique de tous ces modèles est l'unicité des noms des objets présents dans le schéma.

3.2.1 Le modèle Entite-Association[1]

La notion fondamentale du modèle de *[Chen 76]* est l'**entité**: c'est un concept concret (ou abstrait) de la réalité qu'on veut modéliser (par exemple une voiture, un accident, une réparation, . . .). Ce concept est jugé par le concepteur comme ayant une importance suffisante pour être autonome.

Ces entités sont toutes distinctes mais malgré tout, on peut leur trouver des similitudes (propriétés communes, . . .) ce qui fait que l'habitude (et le modèle) veut qu'on les regroupe en classes (appelées **types d'entités**). Une entité (d'après le modèle) ne peut appartenir qu'à une seule classe.

Une des propriétés que peut avoir une entité est le lien qui l'unit à une autre entité (par exemple, le lien entre une voiture et son propriétaire). Dans le modèle Entité/Association (E/A), ces liens sont appelés **associations**. Comme pour les entités, ces associations sont groupées par classes (appelées **types d'associations**).

Chaque lien joue un rôle particulier auprès de l'entité à laquelle il est relié. Dans l'Exemple 3–1, le lien *possession* entre une voiture et son propriétaire jouerait le rôle *possède* auprès de propriétaire et d'*appartient* auprès de voiture.

[1] La description de ce modèle est une synthèse de notes de cours *[Hainaut 89]*.

Les propriétés qui ne sont pas des liens (et ceci est valable pour les entités comme pour les associations) sont appelées dans le modèle E/A des **attributs**. Une partie de ces attributs et des rôles qui sont attachés à un objet (entité ou association) peut servir à l'identifier parmi les objets de la classe à laquelle il appartient.

Une des facilités qu'offre le modèle E/A (et une des raisons de sa grande popularité auprès des concepteurs) est sa représentation graphique des objets de base qu'il possède. Même si ces représentations possèdent de nombreuses variantes (sans parler des extensions que chacun a ajoutées selon ses convenances pour combler les lacunes du modèle), il est facile de s'y adapter car le nombre de concepts n'est pas énorme et si la forme change, le fond lui reste le même (et compréhensible par tout utilisateur de ce modèle).
Nous employons le graphisme suivant: un type d'entités sera représenté par un rectangle et un type d'associations par un hexagone aplati.
L'avantage du graphique est qu'il est plus formel (et donc plus strict) que le langage naturel: c'est une autre raison de son succès.

Si le graphique offre une richesse d'expression importante, et notamment l'ajout de certaines propriétés que doivent vérifier les objets (les plus utilisées sont le nombre de fois qu'une entité doit jouer un rôle dans une association, appelé connectivité en terme du modèle), il ne résout pas tous les problèmes. Dès lors, le concepteur doit compléter son schéma par une liste de propriétés supplémentaires appelées contraintes d'intégrité.

La Figure 3–1 montre une représentation de l'Exemple 3–1 dans ce modèle

Exemple 3–1: Réparation de voitures accidentées

```
Des propriétaires possèdent une ou plusieurs voitures : ces voitures peuvent
avoir des accidents et dès lors doivent subir une réparation.
Un propriétaire est identifié par son nom et possède en plus une adresse.

Une voiture est identifiée par sa plaque minéralogique.

Un accident est identifié par sa date et par la voiture impliquée
dans l'accident.

De même, une réparation est identifiée par une date et la voiture
réparée. Il y a une contrainte qui établit que la date de réparation est
postérieure à celle de l'accident.
```

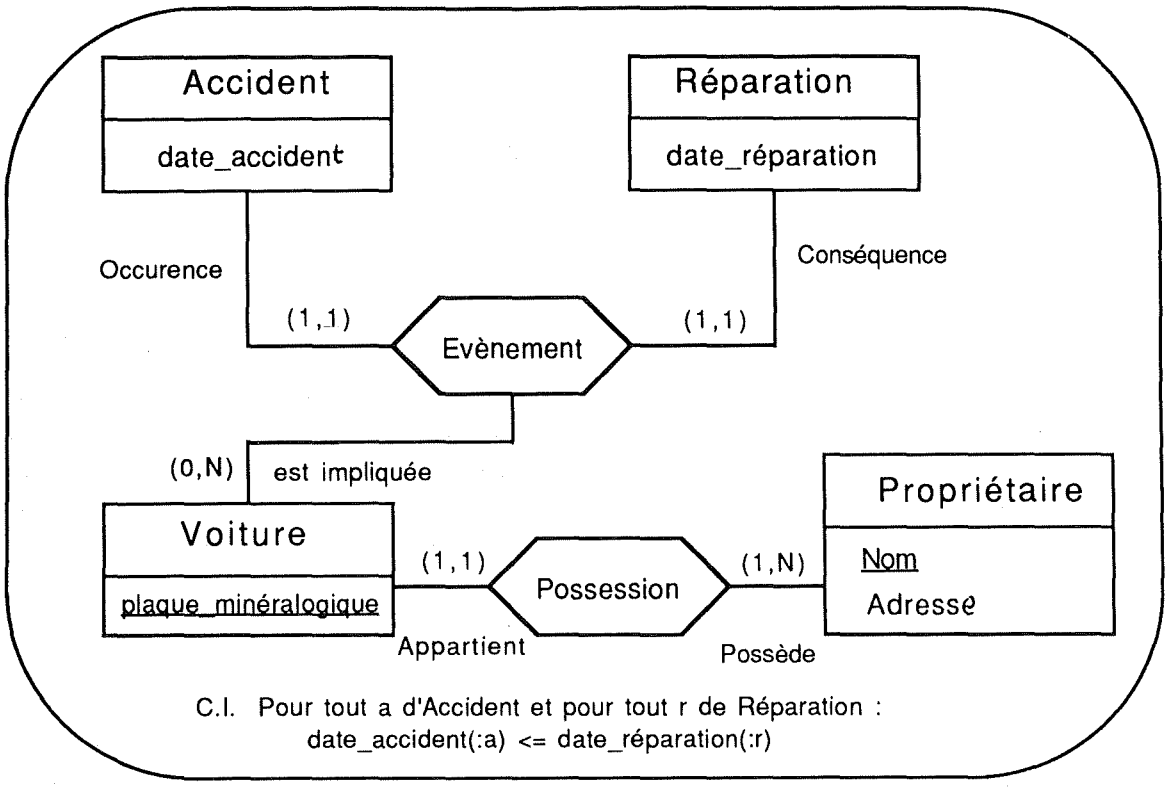
3.2.2 Le modèle Relationnel[2]

Ce modèle a été défini par *[Codd 71]*. L'information représentant des concepts du monde réel se présente sous forme:

- d'ensembles de valeurs de même type appelés **domaines**: chaque valeur de cet ensemble représente un objet ou un évènement (par exemple une voiture, un propriétaire, un accident, . . .);
- d'ensembles d'associations entre valeurs de domaines appelés **relations**. Les associations (ou **lignes**) d'une relation ont la même structure et représentent une association du monde réel;

[2] La description de ce modèle est une synthèse des notes de cours *[Hainaut 88]*.

Figure 3–1: Schéma entité/association



- de diverses **contraintes** portant sur les objets du modèle pour exprimer des propriétés stables de ceux-ci.

L'ensemble des relations constitue la base de données. Une façon graphique de représenter ces relations est sous forme de tableaux (appelés **tables**) où les lignes sont les associations enregistrées et les colonnes sont les domaines (jouant le rôle d'**attribut**) communs à toutes les associations d'une même relation.

A ce formalisme, le modèle ajoute une algèbre relationnelle pour construire un discours structuré et moins ambigu que le langage naturel. Nous retrouvons

- les opérateurs ensemblistes classiques:
  - union,
  - intersection,
  - différence,
  - produit cartésien,
  - complémentaire;
- et des opérateurs plus spécifiques:
  - projection (pour cacher certains attributs d'une relation),
  - jointure (pour fusionner plusieurs relations sur base d'un critère commun),
  - composition (pour coupler deux relations),
  - sélection (pour cacher certaines associations d'une relation), et

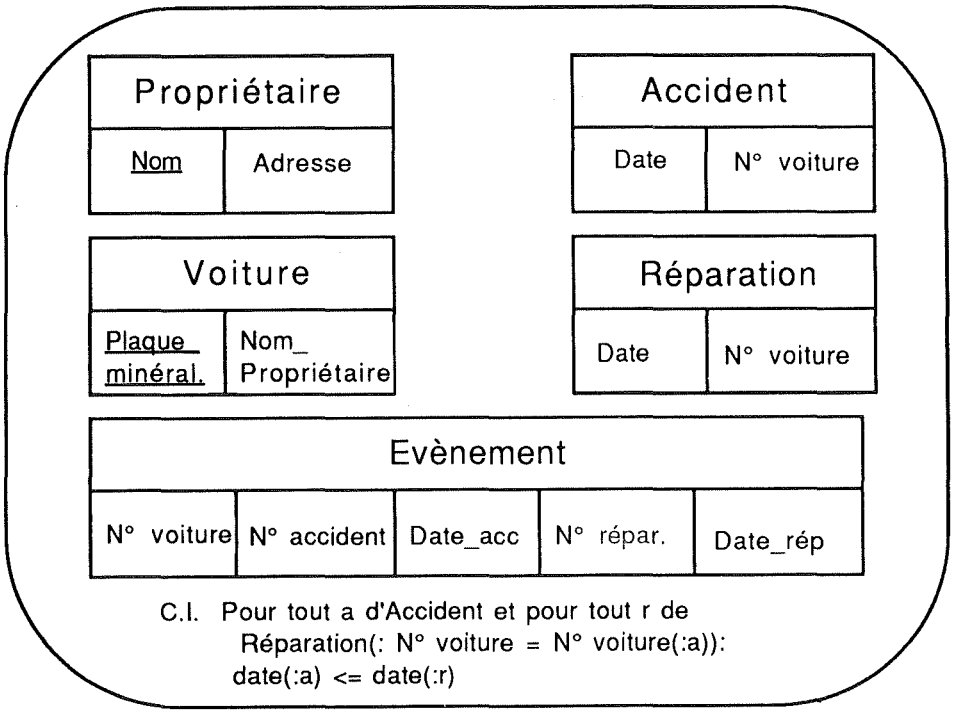
— division (avec un quantificateur universel).

Comme dans le modèle E/A, certains attributs d'une relation peuvent constituer un moyen d'identifier les n-uplets parmi une relation.

Comme le modèle est un système formel, on a développé beaucoup de théorie pour démontrer les assertions et les pratiques de ce modèle: c'est un des avantages majeurs sur les autres modèles. Par contre, un de ses points faibles est qu'il offre peu de moyens pour représenter des structures riches. Ce modèle était idéal pour la gestion dans les années 70 (époque à laquelle il est apparu) mais est beaucoup moins appropriés pour des bases de données graphiques, historiques, . . . vers lesquelles nous nous orientons aujourd'hui.

Reprenons l'Exemple 3–1 pour comparer les différences de représentation entre les modèles relationnels (voir Figure 3–2) et E/A (voir Figure 3–1):

Figure 3–2: Schéma relationnel



3.2.3 Le modèle Fonctionnel

Ce modèle est basé sur deux concepts principaux: les **domaines** et les **fonctions** mais la façon de les utiliser peut différer quelque peu suivant les auteurs.

Un domaine (qui peut être simple ou construit à partir d'autres domaines) représente des objets du monde réel. En général les domaines sont atomiques et représentent des concepts ponctuels (par exemple un nom est un domaine élémentaire dont les valeurs appartiennent à l'ensemble des chaînes de caractères) ou des concepts plus généraux (par exemple un propriétaire est un domaine élémentaire dont les "valeurs" appartiennent à l'ensemble des être humains qui possèdent une voiture).

Un domaine peut aussi être construit sur base de données atomiques, par exemple une adresse est la composition d'un numéro de rue (dont les valeurs sont des entiers), d'un nom de rue et d'une localité (dont les valeurs sont des chaînes de caractères).

Ces domaines sont reliés par des fonctions qui expriment le lien sémantique qui les unit (lien entre un propriétaire et son nom ou bien lien entre une voiture et un accident). Ces fonctions peuvent avoir des utilités différentes: elles peuvent exprimer le fait

— qu'un domaine est attribut d'un autre,

— qu'un domaine est généralisation d'un autre *([Motro 87])*,

— qu'il existe une identité physique entre deux domaines et que ces domaines ne sont en fait que deux représentations distinctes du même objet,

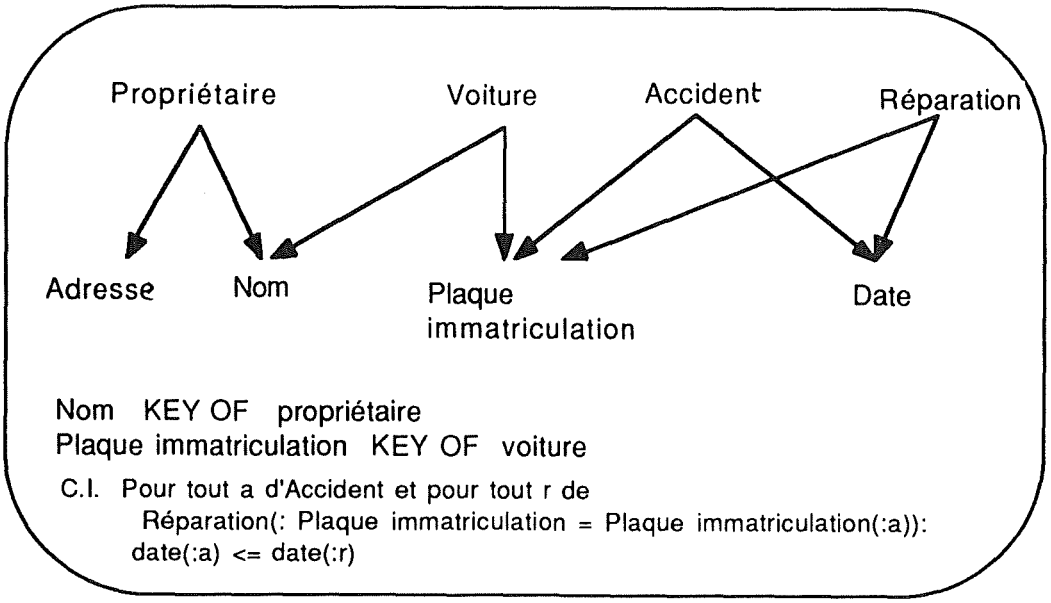
.
.
.

Ce modèle aussi peut être complété par des contraintes exprimant les propriétés de la réalité qui n'ont pu être modélisées avec les concepts offerts par le modèle.

L'avantage de ce modèle, c'est qu'il offre une grande facilité pour traiter des objets identiques mais dont les représentations sont différentes. L'inconvénient est qu'il est un peu difficile à comprendre.

La Figure 3–3 montre la représentation graphique du modèle fonctionnel pour l'Exemple 3–1.

Figure 3–3: Schéma fonctionnel



3.3 Les stratégies d'intégration

Parmi les différentes démarches analysées, nous pouvons distinguer plusieurs solutions quant au nombre de schémas à prendre en compte lors d'une étape d'intégration. Nous trouvons deux stratégies (elles-mêmes subdivisées en deux catégories):

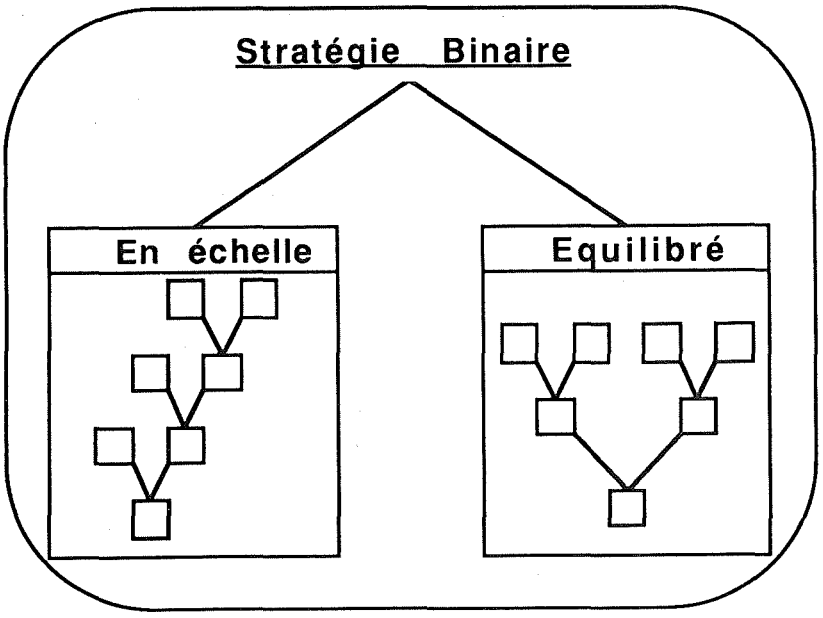
3.3.1 La stratégie binaire

Elle correspond à prendre pour chaque pas d'intégration, deux schémas en entrée du processus et à les traiter entièrement (i.e. réalisation de toutes les étapes de l'intégration) avant de continuer le processus d'intégration avec d'autres schémas.

Cette stratégie fait l'objet de deux variantes (voir Figure 3–4):

- le mode **équilibré** correspond à choisir des schémas non encore traités et à leur faire franchir un pas d'intégration. Une fois que tous ont franchi ce pas (ou qu'il n'en reste plus qu'un si le nombre de schéma à intégrer est impair), on reprend les schémas résultats et on leur fait franchir un nouveau pas;
- l'intégration **en échelle** correspond à choisir deux schémas de départ quelconques. Une fois le schéma résultat obtenu, on reprend celui-ci et on choisit un autre schéma parmi les schémas restant à intégrer, pour effectuer le pas suivant de l'intégration.

Figure 3–4: La stratégie binaire



L'avantage de cette stratégie est qu'elle simplifie les activités de comparaisons des objets dans les schémas à traiter. Nous pouvons montrer qu'un algorithme de fusion est de complexité n^2. Il est intéressant, si nous voulons minimiser la complexité, de prendre "n" le plus petit possible.

L'inconvénient de cette stratégie est qu'il faut répéter l'étape d'intégration plusieurs fois (puisqu'on choisit de découper au maximum l'ensemble des schémas à intégrer pour minimiser la complexité). Cela implique qu'une fois le processus terminé, il faut faire une analyse globale du résultat obtenu pour ajouter des propriétés qui portaient sur plusieurs (au moins trois) schémas et qu'on n'a pas pu prendre en considération lors des étapes précédentes (vu qu'on se limitait à deux schémas). Il faut aussi détecter les cycles redondants et restructurer le schéma intégré pour qu'il soit plus clair et compréhensible.

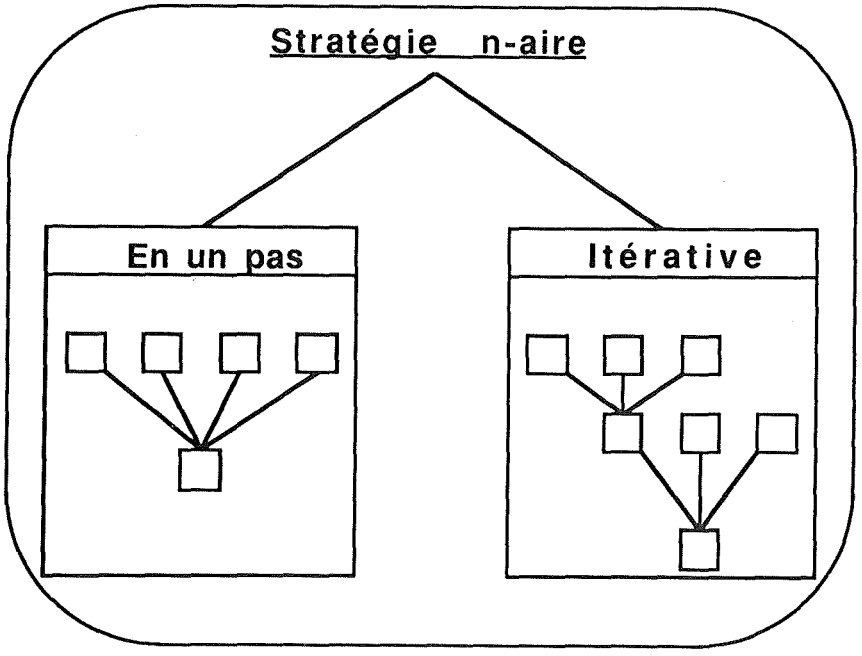
3.3.2 La stratégie n-aire[3]

Elle correspond à prendre tous les schémas qu'il faut intégrer et à réaliser leur fusion (après résolution des conflits) en un seul pas.

Nous pouvons, de nouveau, observer deux variantes de cette stratégie (voir Figure 3–5):

- l'intégration **en un pas** correspond à la philosophie de cette stratégie: on analyse tous les schémas simultanément. On obtient ainsi un résultat intégré en une phase d'intégration. Cela implique un mécanisme d'analyse et de résolution de conflits très complexe;
- le mode **itératif** correspond à adopter une position intermédiaire entre la stratégie binaire et n-aire pure: on prend un sous-ensemble des schémas à intégrer pour simplifier les étapes d'analyse et de résolution de conflits. Il faut par cette politique, procéder en plusieurs phases d'intégration (comme dans la stratégie binaire).

Figure 3–5: La stratégie n-aire



[3] Il faut que "n" soit strictement supérieur à 1.

Les avantages de la solution n-aire sont:

— puisqu'on a tous les schémas à sa disposition, on peut détecter des propriétés inter-schémas[4]. Une analyse du résultat obtenu après l'intégration n'est plus nécessaire;

— le nombre de pas est minimisé.

L'inconvénient, comme déjà exposé plus haut, est la complexité accrue dans l'étude et la résolution des conflits par rapport à la stratégie binaire.

3.4 Les écoles d'intégration

Dans cette section, nous présentons les différentes écoles qui nous sont apparues lors de notre recherche de techniques et de méthodologies d'intégration. Nous avons construit cette classification sur base de concepts et de ressemblances que nous avons trouvés dans les différents articles.
Il se peut que d'autres auteurs adoptent une classification différente.

Notre classification reprend uniquement les principales écoles de pensée en matière d'intégration.

3.4.1 Introduction

Une école d'intégration se définit par une politique d'intégration adoptée par des chercheurs. Chaque idée est potentiellement apte à créer une école et si cette idée est prometteuse, elle est reprise par d'autres: quand un certain nombre de chercheurs ont repris cette idée, nous pouvons dire que nous sommes en présence d'une école d'intégration.

Chaque école contient des similitudes avec les autres quant à la technique pour produire une base de données globale contenant les informations des différents schémas locaux. Pour l'utilisateur, le résultat final est toujours le même: il a l'impression d'avoir accès à toutes les données en un seul endroit. Cependant, il existe des différences suivant les objectifs que l'on veut poursuivre[5].

La différence sur laquelle nous nous sommes basés pour faire cette classification est le moment choisi pour faire l'intégration.

Notre classification reprend quatre types d'école de pensée:

1. l'école d'intégration dynamique,
2. l'école d'intégration virtuelle partielle,
3. l'école d'intégration virtuelle globale, et
4. l'école d'intégration physique globale.

[4] Si on ne prend pas tous les schémas, il suffit d'en prendre n, ceux sur lesquels porte la propriété.

[5] Une de ces différences, mais qui ne nous paraît pas fondamentale pour classifier les écoles, est le choix du modèle sous-jacent au schéma global: chacun utilise le schéma qui correspond le mieux à ses objectifs.

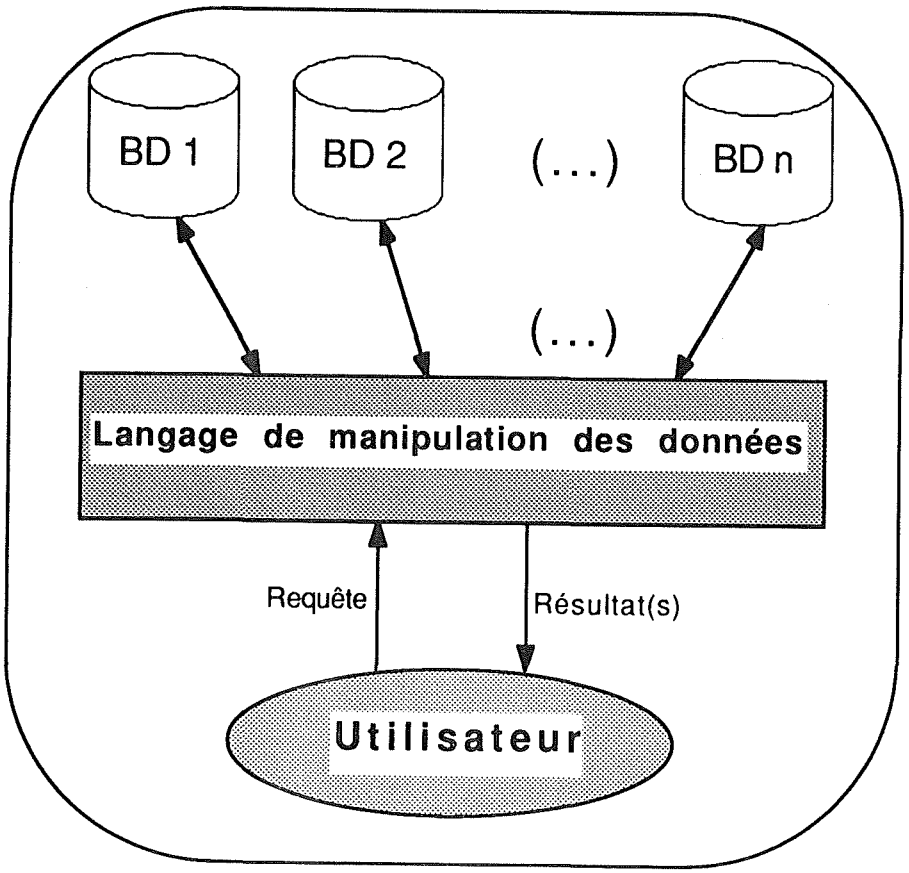
3.4.2 Intégration dynamique

Cette école adopte comme politique d'intégrer à chaque requête les données nécessaires pour la construction de la réponse.

Le but de cette école est de garder distribuées toutes les bases de données locales sans être forcé de fusionner toutes les bases de données en un seul schéma global. Pour réaliser cet objectif, l'école a développé un langage de manipulation de données qui peut accéder à plusieurs bases de données à la fois. L'interpréteur du langage doit réaliser une intégration temporaire (cette intégration n'est pas physiquement visible) pour répondre aux requêtes.

Le langage de manipulation des données prend la requête introduite par l'utilisateur et adresse aux bases de données (pas nécessairement toutes) des sous-requêtes pour répondre à la demande de l'utilisateur. Un fois les résultats obtenus, le langage intègre les données et renvoie le résultat vers l'utilisateur (voir Figure 3–6).

Figure 3–6: Schéma de l'intégration dynamique



Un exemple d'un tel langage d'intégration est celui utilisé par *[Litwin 84]*: le langage MALPHA. Ce langage permet la manipulation de collections de bases de données relationnelles. Les requêtes MALPHA peuvent accéder aux données de bases de données différentes et les combiner pour former une seule réponse.
Le modèle de données utilisé par ce langage est le modèle relationnel. Tous les schémas locaux doivent avoir été écrits dans ce modèle.

L'avantage de cette école est de garder tous les schémas locaux dans leur base de données d'origine.
Les inconvénients sont que:

— cette école ne permet pas, comme nous l'avons vu pour l'exemple du langage MALPHA, d'accéder à des schémas de données hétérogènes,

— chaque fois qu'une requête est traitée, l'intégration doit être reconstruite: si c'est une facilité pour détecter les modifications entre deux requêtes, cela peut s'avérer coûteux en temps d'exécution.

3.4.3 Intégration virtuelle partielle

Cette école adopte comme politique d'intégrer des bases de données existantes. Puisqu'elles existent déjà, il est quasiment impossible de créer une nouvelle base qui reprend de façon complète et cohérente toutes les données de ces bases. Pour contourner cet obstacle majeur, on ne crée qu'une interface virtuelle.
L'intégration accomplie est partielle: elle ne réalise que le minimum souhaité par l'utilisateur.

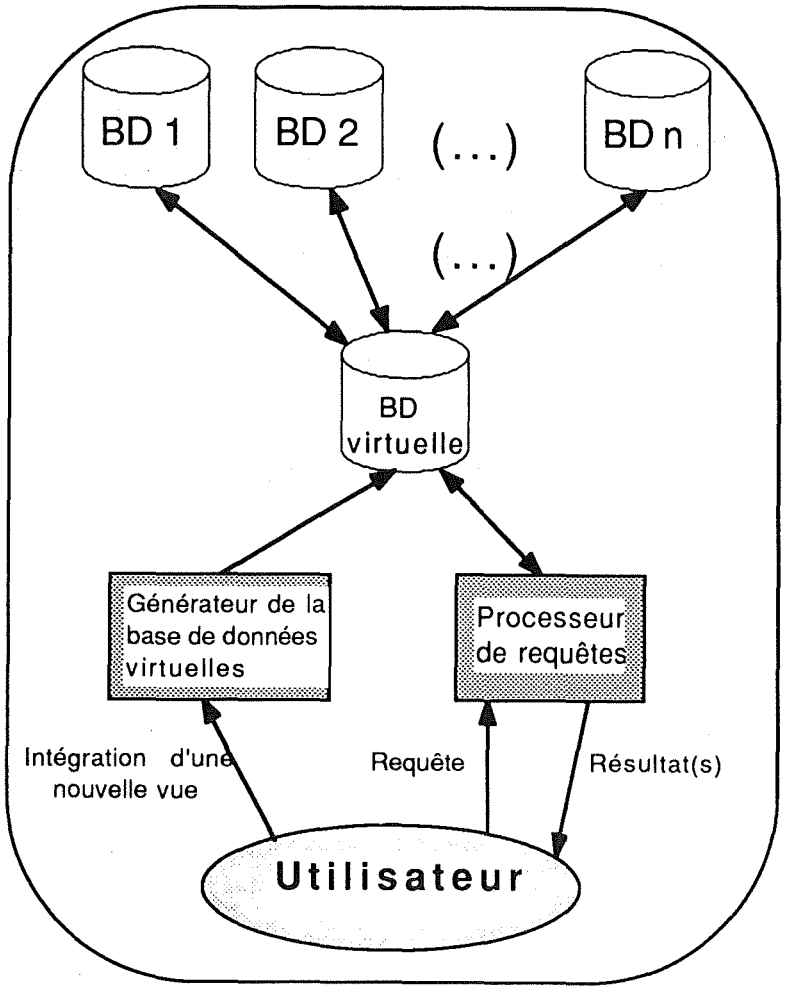
Le but de cet école est d'optimiser la méthode proposée par l'école précédente en construisant des vues virtuelles qui mémorisent seulement les chemins d'accès aux données pour éviter d'avoir à les reconstruire à chaque requête.

L'utilisateur peut soit interroger le processeur de requêtes. Celui-ci réalise l'interface entre l'utilisateur et la base de données virtuelle. Cette base contient les indirections nécessaires vers les bases de données locales peut répondre à la requête de l'utilisateur.
Soit l'utilisateur peut intégrer une nouvelle vue dans la base de données virtuelle. Pour cela, il donne au générateur de la base de données virtuelle, les ordres nécessaire pour incorporer la nouvelle vue dans la base de données virtuelle (voir Figure 3–7).

Le modèle le mieux adapté à cette école semble être un modèle fonctionnel avec, comme dans tout modèle de ce type, les notions de domaine (ensemble de valeurs) et de fonctions (entre ces domaines). L'Exemple 3–2 nous montre le modèle fonctionnel proposé par *[Motro 87]*.

Pour intégrer, cette école développe un langage qui est implémenté avec un petit nombre d'opérateurs d'accès.

Figure 3–7: Schéma de l'intégration virtuelle partielle



Exemple 3–2: Le modèle d'intégration virtuelle de *[Motro 87]*

```
L'auteur décrit une base de données comme un ensemble de classes ayant entre
autre un domaine. Ses opérateurs d'accès sont définis sur des classes. Il
propose comme opérateurs:
- domain (class),
- function (class), et
- inverse (class).

La signification de ces opérateurs a été définie dans la section 2.7.2.
```

Puisque l'un des objectifs de cette école est d'intégrer le minimum nécessaire à un utilisateur (pour être plus rapidement opérationnel car une intégration est un processus qui prend du temps), comme le générateur de base de données virtuelle ne sait pas ce que souhaite l'utilisateur[6], il est normal que ce soit l'utilisateur qui fasse l'intégration. Cette politique résout le problème du conflit de noms en le laissant à l'utilisateur (qui est le mieux à même de connaître la signification des objets qu'il manipule).

[6] Il pourrait le demander mais comme le processus d'intégration est interactif, cela surchargerait l'intégration. Puisqu'il est obligé de demander quelque chose à l'utilisateur, il faut que ce soit le minimum.

Pour faire cette intégration, cette école fournit des outils qui modifient les chemins d'accès aux données mémorisés par les vues virtuelles pour répercuter les modifications introduites par l'utilisateur et ajouter les nouvelles contraintes apportées par les transformations de l'utilisateur. Toutes ces informations sont mémorisées dans de nouvelles vues virtuelles qui peuvent être réutilisées par la suite.

Les requêtes sont traitées en commençant par les vues les plus récentes et en remontant (en suivant les chemins indiqués dans ces vues) jusqu'aux données. Les requêtes sont construites en termes des opérateurs d'accès (qui peuvent être combinés pour former des requêtes complexes).

Les avantages de cette école sont:

— de garder les schémas d'origine indépendants,

— d'intégrer seulement les parties nécessaires aux traitements des requêtes, et

— de réutiliser les descriptions virtuelles déjà existantes (et donc d'être plus rapide que la première école).

L'inconvénient est que comme l'intégration n'est pas globale et que les données d'origine sont conservées, certaines données peuvent être contradictoires et la requête ne peut être traitée. Il faut que l'école adopte une politique qui réduise les inconvénients de ce genre.

3.4.4 Intégration virtuelle globale

Cette école a pour politique l'intégration des bases de données existantes. Puisqu'elles existent déjà, il est quasiment impossible de créer une nouvelle base qui reprend de façon complète et cohérente toutes les données de ces bases. Pour contourner cet obstacle majeur, on ne crée qu'une interface virtuelle.

Le but de cette école est d'offrir une interface globale pour éviter des conflits entre données. Cette politique d'empêcher les conflits entre les données de schémas compatibles peut être étendue à des schémas hétérogènes.

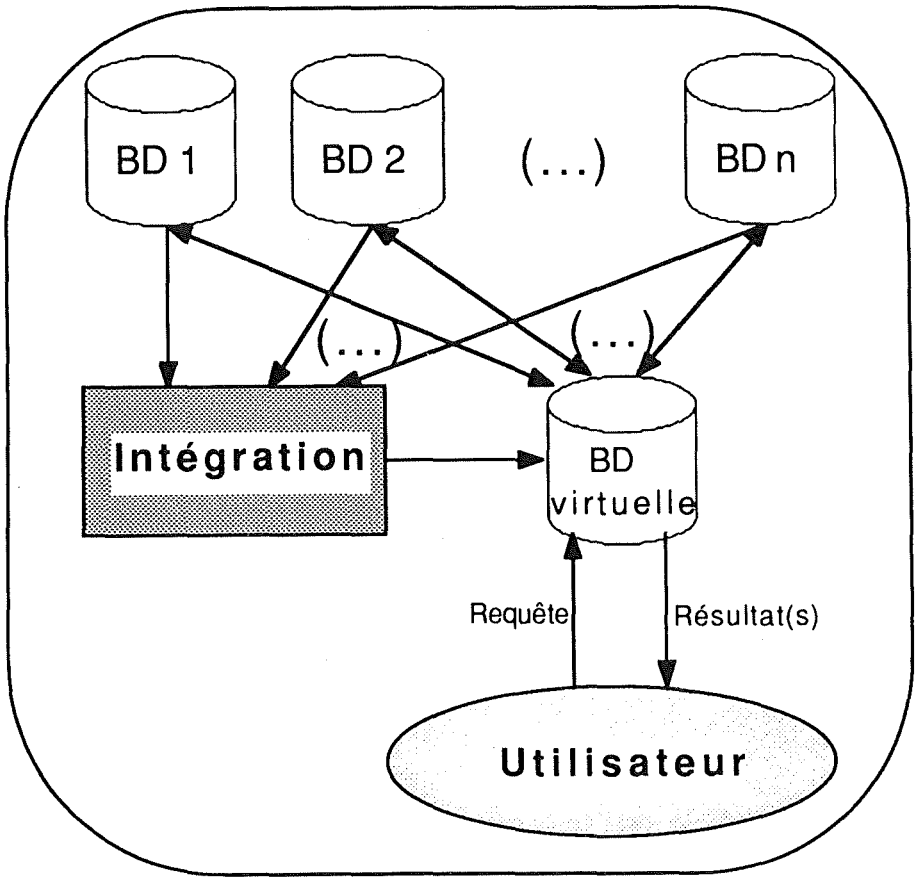
La Figure 3–8 montre les étapes du processus d'intégration dans cette école:

1. dans un premier temps, on intègre les différents schémas locaux pour obtenir une base de données virtuelle.
2. Après quoi, l'utilisateur emploie cette base de données virtuelle comme interface dans la définition de ses requêtes vers les différentes bases de données locales.

L'école ne touche pas aux schémas locaux, permettant ainsi de continuer à utiliser les applications qui tournent sur ces schémas. Elle copie la sémantique de chacune de ces vues et la formalise en un modèle unique (celui qu'utilise l'intégrateur). C'est sur ces traductions de schémas que sont faites les transformations pour aboutir à une interface globale virtuelle. Quand l'utilisateur introduit une requête, il croit qu'il travaille sur une base de données unique. En fait, c'est l'interface qui traite la requête et se charge de récolter dans les différentes bases les éléments de réponse.

Le modèle le plus souvent utilisé est le modèle fonctionnel: il est le mieux adapté à "aplanir" les différences qui existent entre les diverses bases.

Figure 3–8: Schéma de l'intégration virtuelle globale



Les avantages sont les mêmes que ceux de l'intégration virtuelle partielle.

3.4.5 Intégration physique globale

Cette école a pour politique l'intégration des schémas de bases de données en cours de conception. Cette hypothèse permet d'optimiser les schémas car les répercussions sont moindres (notamment par le fait qu'il n'y a pas encore de données introduites dans la base).

Le but de cette école est de fournir un schéma intégré global qui préserve la sémantique de tous les schémas d'origine tout en empêchant des inconsistances, de la redondance, ...

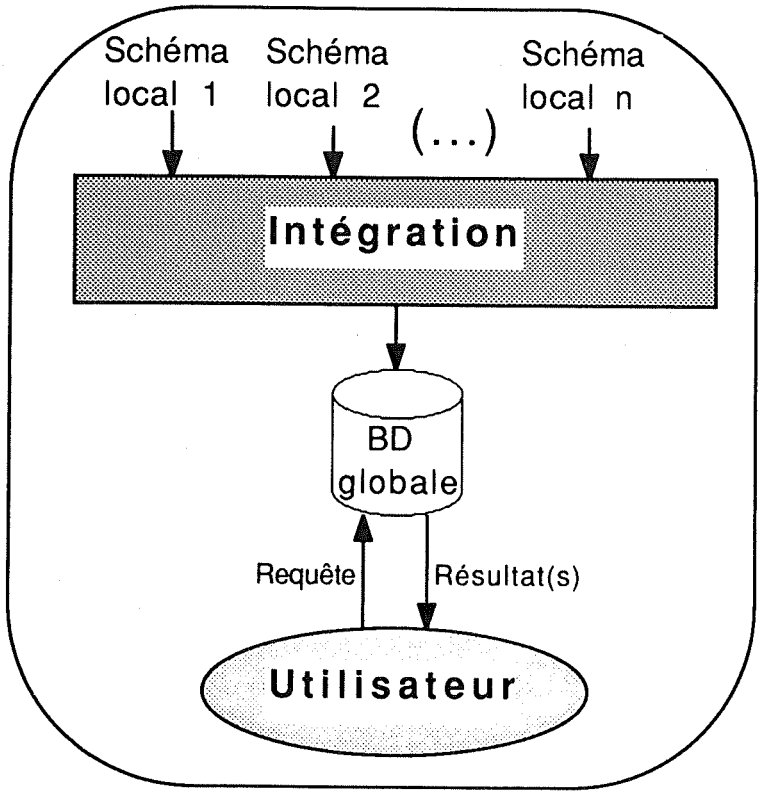
C'est dans cette école que nous trouvons le plus de variantes dans le choix du modèle de données qui supportera la technique d'intégration.

Cette école est aussi celle qui compte le plus de partisans et il est normal de voir une foule de variantes (intégration en une étape ou par une itération d'intégrations de deux vues, . . .). Le résultat final est l'obtention d'un seul schéma global. C'est dans celui-ci que seront introduites les données. Cette école peut (et c'est d'ailleurs l'évolution que nous constatons) conserver les schémas d'origine: le processus d'intégration fournira à chaque utilisateur une interface qui présentera les données de la manière qu'il souhaite, mais elles seront stockées dans une base de données centrale.

La Figure 3–9 montre les étapes du processus d'intégration dans cette école:

1. dans un premier temps, on intègre les différents schémas conceptuels pour obtenir un schéma global.
2. Après quoi, il faut créer la base de données physique respectant ce schéma global obtenu. L'utilisateur posera alors ses requêtes à cette base de données globale.

Figure 3–9: Schéma de l'intégration physique globale



Les avantages d'une telle école sont que:

— le résultat obtenu est complet, cohérent et sans redondance[7], ce qui n'est pas toujours le cas dans les trois autres écoles;

— le résultat peut être optimisé car on n'est pas contraint à respecter un certain formalisme puisque rien n'existe encore: si on peut garantir l'équivalence sémantique, les concepteurs opteront pour la solution la plus rapide.

L'inconvénient est que cette méthode est difficilement applicable si les bases de données existent déjà.

[7] La redondance implique en effet un surcoût de gestion et de complexité lors de la modification d'une donnée dupliquée.

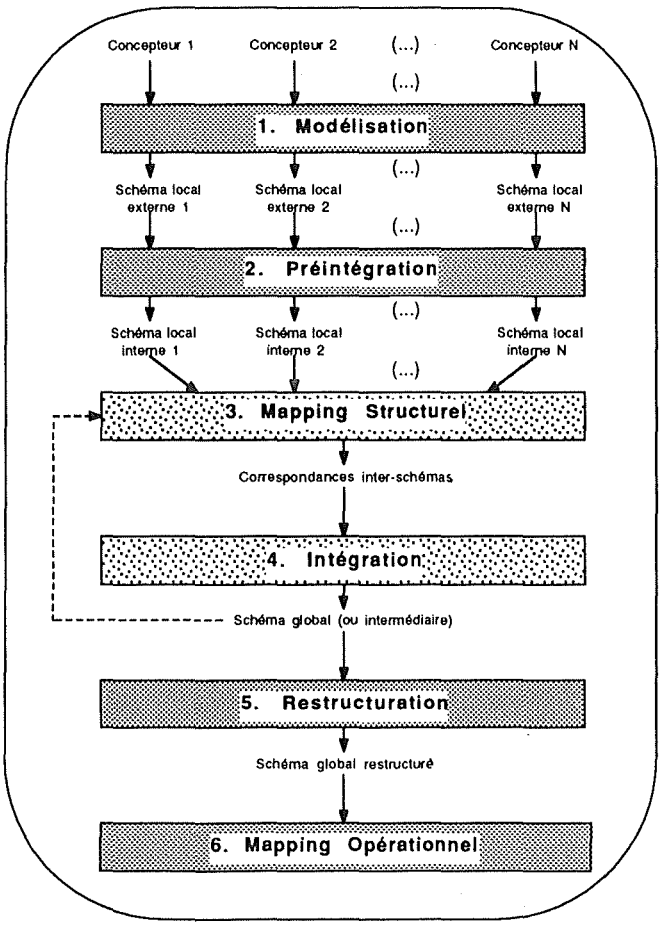
3.5 Démarche générale du processus d'intégration

Nous observons lors de la description des différentes démarches d'intégration au Chapitre 2, que chacune d'entre elles suit une démarche d'intégration qui lui est propre.

Nous pouvons sur base de l'analyse de ces méthodologies construire une démarche générique du processus d'intégration (voir Figure 3–10) qui est un mélange des activités suivantes:

1. modélisation,
2. préintégration,
3. mapping structurel,
4. intégration,
5. restructuration,
6. mapping opérationnel.

Figure 3–10: Démarche générale du processus d'Intégration



3.5.1 Modélisation

Lors de cette étape, chaque concepteur modélise (et indépendamment des autres concepteurs), sur base des spécifications reçues, l'univers du discours au moyen des concepts d'un modèle de structuration des données qu'il s'est choisi.

On obtient, à la fin de cette étape, un schéma conceptuel respectant les spécifications du problème à résoudre, ainsi que les conventions du modèle de données dans lequel le schéma a été modélisé.

3.5.2 Préintégration

Cette étape a pour objectif de préparer le cadre de travail du processus d'intégration par l'analyse les différents schémas à intégrer.

Cette analyse comprend entre autre:

— le choix d'une politique d'intégration des schémas,

— l'ordre d'intégration par assignation d'un poids à chaque schéma (par exemple en fonction de la taille des schémas), et

— la préférence d'intégrer un schéma dans son ensemble ou seulement sur base d'une portion de celui-ci.

Cette étape s'occupe aussi de la conversion des schémas en un formalisme identique ou dans le formalisme interne du processus d'intégration (par exemple convertir tous les schémas dans le modèle Entité/Association de *[Chen 76]*). Cela permet de ne pas imposer des contraintes aux concepteurs quant au modèle à utiliser.

De plus, c'est lors de cette étape qu'on prend la décision sur la stratégie d'intégration, ainsi que sur le taux d'interactivité entre le processus d'intégration et l'administrateur de base de données.

La collection d'informations complémentaires, telles que des affirmations, propriétés ou contraintes des différentes composantes des vues, est aussi prise en considération lors de cette phase. Dans la méthodologie de *[Larson & Al 89]* par exemple, les concepteurs avant d'intégrer leur schéma avec les autres doivent donner une description détaillée des différents attributs.

3.5.3 Mapping structurel

Lors de cette étape, on analyse les schémas pour détecter les éventuels conflits.
On distingue deux catégories de conflits:

CONFLITS DE TERMINOLOGIE: ce sont les conflits de noms entre les concepts des différents schémas. On peut avoir des concepts synonymes entre eux: concepts qui représentent la même chose mais qui portent des noms différents, et des concepts homonymes entre eux: concepts qui ont le même nom mais ne représentent pas le même monde réel.
CONFLITS DE STRUCTURES: le même monde réel est représenté par des concepts différents dans les schémas à intégrer. Par exemple dans le modèle Entité/Association de *[Chen 76]*, on peut avoir les conflits structurels suivants:

- entité/association;

- entité/attribut;
- association/attribut.

On élabore aussi le mapping structurel: c'est un ensemble contenant toutes les correspondances inter-schémas entre les différents schémas à intégrer.

La typologie de correspondances inter-schémas est la suivante:

Correspondance d'identité

Les différents concepteurs modélisent un même ensemble de propriétés d'une même classe d'objets du monde réel par des structures identiques.

Eventuellement le nom des objets peut être différent, on a affaire alors à des concepts synonymes[8].

Correspondance d'équivalence

Les différents concepteurs modélisent un même ensemble de propriétés d'une même classe d'objets du monde réel par des structures différentes.

Correspondance d'inclusion

Les concepteurs modélisent un ensemble de propriétés d'une même classe d'objets du monde réel avec des structures éventuellement différentes. Les différents ensembles de propriétés sont inclus l'un dans l'autre.

Correspondance de recouvrement

Les concepteurs modélisent un ensemble de propriétés d'une même classe d'objets du monde réel avec des structures éventuellement différentes. Les différents ensembles de propriétés ne sont ni équivalents, ni inclus l'un dans l'autre.

Correspondance d'exclusion

Les concepteurs modélisent un ensemble différent de propriétés de différentes classes d'objets du monde réel avec des structures éventuellement différentes. Les différents ensembles de propriétés ne sont ni équivalents, ni inclus , ni recouverts l'un dans l'autre.

3.5.4 Intégration

Une fois la détection des conflits et le mapping structurel réalisés, il faut intégrer l'ensemble des schémas sur base du mapping structurel et en résolvant les conflits entre les concepts des différents schémas.

On obtient à la fin de cette étape un schéma global, qui est l'intégration des différents schémas locaux.

En fonction de la stratégie d'intégration adoptée à la phase de préintégration (section 3.5.2), il se peut que l'on doive itérer plusieurs fois la phase 3 (section 3.5.3) et 4 (section 3.5.4), avant de passer à l'étape suivante, qu'est la restructuration du schéma global(section 3.5.5).

[8] On peut constater qu'une gestion des concepts synonymes et homonymes par un outil peut représenter une aide efficace dans l'élaboration des correspondances inter-schémas.

3.5.5 Restructuration

A ce moment, l'intégration proprement dite des différents schémas est accomplie. Il se peut que le résultat obtenu ne vérifie pas un certain nombre de critères de qualité. Le schéma global peut être testé sur les critères de qualité suivants:

- **Complétude et Cohérence.** Le schéma intégré doit contenir tous les concepts présents dans les différents schémas locaux. Le schéma obtenu à la fin de la phase d'intégration doit être une représentation de l'union des classes d'objets du monde réel représentées dans les schémas locaux.
- **Minimalité.** Si un même concept apparaît dans plus d'un schéma local, il ne peut figurer qu'une seule fois dans le schéma global.
- **Compréhensivité.** Le schéma global doit être facile à comprendre pour le concepteur et l'utilisateur final. Cela implique que parmi les différentes possibilités de représentations qu'offrirait le modèle de données, on choisit la représentation la plus compréhensible.

3.5.6 Mapping opérationnel

Le **mapping opérationnel** est l'ensemble des règles qui sélectionnent les parties du schéma global ainsi que les occurrences de manière à satisfaire une requête exprimé sur un schéma local.

3.6 Les types de démarches

Nous entendons par démarche la façon dont se déroule le processus d'intégration et les actions que doit poser la personne qui intègre (e.g. l'administrateur des bases de données). Nous pouvons classifier les démarches en fonction de la participation d'un outil à ce processus. Il y a trois cas possibles:

1. l'outil n'intervient pas, si ce n'est pour donner éventuellement des indications. Nous disons alors que la démarche est entièrement **manuelle,**
2. l'outil réalise l'intégration en interaction avec le concepteur. Nous disons alors que la démarche est **interactive,**
3. l'outil effectue l'intégration sans aucune interaction avec le concepteur. Nous disons alors que la démarche est **automatique.**

3.6.1 Démarche manuelle

Dans ce cas, c'est la personne chargée de faire l'intégration qui doit repérer les similitudes entre schémas et éliminer les conflits de structures, de noms, ... Pour ce faire, elle se base sur les spécifications fournies par les différents concepteurs (et peut, si besoin est, leur demander des informations complémentaires) et les indications faites par des outils d'aide à l'intégration (détection des conflits, des correspondances, . . .).

Une fois que l'intégrateur maîtrise bien les données du problème, il peut passer à la phase d'intégration et fusionner les différents schémas pour obtenir un schéma intégré qu'il devra écrire de toutes pièces. Comme l'intégrateur passe en revue tous les éléments du problème, il peut facilement agréger des éléments et faire les optimisations qu'il juge adéquates.

Le problème principal que pose l'intégration manuelle est que la personne qui intègre doit être très compétente pour pouvoir traiter des schémas écrits avec des formalismes différents. Malgré toute la compétence qu'il possède, il n'est pas à l'abri d'oubli dès que les schémas à intégrer sont trop importants en taille et en nombre.

3.6.2 Démarche Interactive

Nous qualifions aussi cette démarche d'**assistée**, car l'outil y joue un rôle important: en effet, il analyse les schémas en entrée et peut faire déjà un premier filtrages des problèmes. L'outil peut en effet détecter les situations problématiques et les rapporter à la personne chargée de l'intégration. Celle-ci n'a "plus qu'à" décider quelle solution semble la plus avantageuse.

Après la détection des problèmes, l'outil peut prendre en charge l'intégration et la vérification de la cohérence du mapping structurel. Cet outil, lors de l'intégration, dialogue avec le concepteur pour que celui-ci choisisse une solution parmi celles proposées par l'outil pour résoudre l'intégration des différents conflits.

3.6.3 Démarche automatique

Il est illusoire de croire que cette démarche d'intégration peut être totalement automatique. Il y a un tas de problèmes qui ne sont pas solubles en général mais doivent être traité au cas par cas. Cet état de fait rend impossible une démarche entièrement automatique mais certaines parties du processus d'intégration peuvent être totalement automatisée.

Parmi ces parties, nous trouvons les conflits de noms qui peuvent être éliminés en renommant de façon unique tous les objets en conflits. Cette solution n'est certainement pas idéale car un conflit de nom peut être un indice pour retrouver des éléments qui pourraient avoir la même sémantique.

Une autre partie est la phase d'intégration. Pour cela, il faut que l'intégrateur ait auparavant décrit de façon complète et précise toutes les correspondances structurelles qui existent entre les éléments des schémas à intégrer. Avec cette description, l'outil peut réaliser les transformations nécessaires et éliminer les problèmes en prenant les solutions les plus probables sur base des indices qu'il a sa disposition. Un exemple de ce type de démarche est celle proposée par *[Spaccapietra & Parent 90]*.

Puisqu'il lui faut une description complète (ce qui prouve bien que la démarche n'est pas entièrement automatique), on peut fournir des outils automatiques pour détecter si la description est effectivement cohérente.

3.7 Les problèmes de l'intégration

Nous avons déjà rencontré au cours de ce chapitre, lors de la présentation des différents concepts que cache un processus d'intégration, quelques problèmes que pose l'intégration de schémas. Nous allons présenter maintenant la liste des principaux problèmes que nous avons pu répertorier chez les auteurs présentés au Chapitre 2, ETAT DE L'ART SUR L'INTEGRATION DES VUES.

3.7.1 Ordre d'intégration

Puisqu'il y a plusieurs schémas à intégrer (sinon il n'y a pas d'intégration), la personne chargée de cette opération doit décider par quel schéma commencer pour faire l'intégration (surtout dans une stratégie binaire). Ce choix peut s'avérer capital pour l'obtention du schéma final. Lorsqu'arrive un problème d'incompatibilité entre deux structures, l'ordre d'intégration adopté influencera la décision quant à la structure à garder. Il est vraisemblable que l'intégrateur écartera la structure la plus récente[9], sur laquelle on a le moins travaillé (pour éviter de perdre un tas de modifications). Cependant, si on adopte une stratégie n-aire, ce problème est singulièrement réduit (voire éliminé) puisqu'on peut aller jusqu'à prendre tous les schémas pour les traiter en une seule étape.

Même lorsqu'il n'y a que deux schémas, l'ordre peut avoir de l'importance car c'est dans le schéma primaire (celui auquel on donne plus d'importance) que se font les transformations[10]. Le choix peut s'avérer capital car les transformations nécessaires pour intégrer sont généralement élémentaires si on enrichit un schéma avec des concepts plus spécialisés vers un schéma agrégé tandis qu'un tas de problèmes surgissent pour intégrer dans l'autre sens (du plus général vers le plus détaillé).
Dans l'exemple de *[Spaccapietra & Parent 90]* en effet, illustré par la Figure 3–11, si on intègre du schéma Ss vers le schéma Sp, il n'y a que des adaptations mineures à faire car tous les objets ont leur correspondant. Par contre dans l'autre sens, il faut créer une entité et tous les objets qui gravitent autour.

Le problème de l'ordre de prise en compte des schémas se pose avec une acuité différente suivant la technique d'intégration adoptée. Cependant, dans les cas où ce problème se pose, c'est un de ceux qui doit être traité en priorité car sans schéma, il n'y a pas d'intégration possible. C'est pourquoi, ce problème est répertorié dans la phase de *préintégration* (*[Batini & Al 86]*).

3.7.2 Problèmes de terminologie

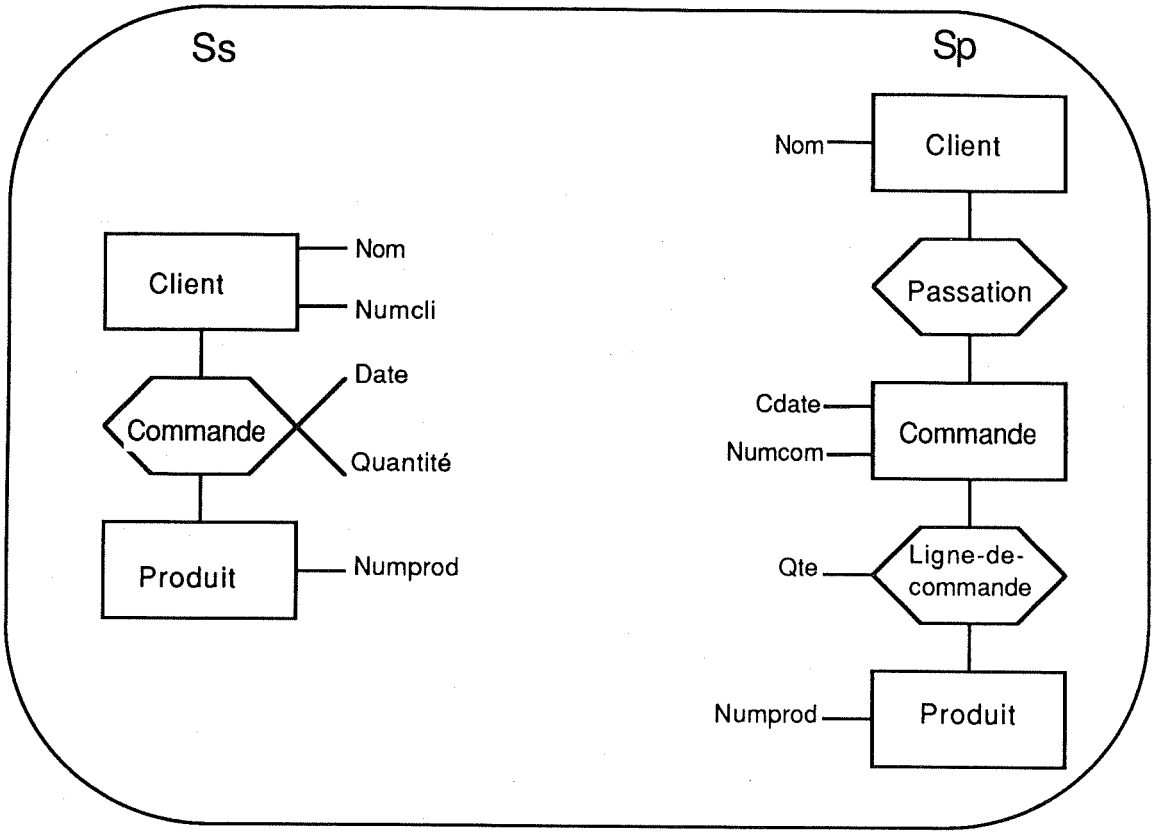
Les problèmes de terminologie (aussi appelés conflits de noms), se posent quand des objets de schémas différents ont le même nom: comme un schéma, quel que soit le modèle adopté ne peut avoir que des (classes d')objets avec des noms uniques, les objets litigieux devront être renommés.

Le changement de nom en lui-même ne pose pas de problème, le tout est de savoir quand le faire car l'équivalence des noms peut indiquer une correspondance entre deux objets: soit qu'ils modélisent la même réalité, soit que les concepts qu'ils représentent se recouvrent (par exemple dans le modèle Entité/Association, une entité appelée voiture pourrait dans un schéma représenter les voitures de la province de Namur et dans l'autre les voitures louées de Belgique).
Dès lors, supprimer cette équivalence de noms avant d'avoir intégré ces deux objets pourrait rendre la phase d'intégration plus difficile car il faudrait repérer la correspondance parmi une liste d'objets (éventuellement importante dans de grands schémas) sans plus avoir

[9] Cet ordre peut être basé, par exemple (*[Batini & Lenzerini 84]*), sur un poids assigné à chaque schéma avant d'entamer le processus d'intégration. Les schémas seront traités selon l'ordre décroissant de leur poids. Ce poids peut correspondre à la fiabilité qu'on accorde au concepteur, la priorité qu'on accorde au schéma en fonction de la politique de l'entreprise, ...

[10] Il existe cependant une autre stratégie qui consiste à créer un troisième schéma en prenant les objets les plus généraux, dès lors il n'y a pas de schéma modifié.

Figure 3–11: Importance de l'ordre d'intégration



l'indice fourni par cette équivalence. Il faudrait consulter pour chaque objet sa description fournie en annexe du schéma, ce qui peut introduire le risque d'oublier une correspondance.

Une équivalence de noms n'implique cependant pas que les objets représentent la même réalité: il se peut tout aussi bien que les concepteurs des schémas impliqués dans le conflit aient choisi les mêmes noms par hasard[11].
Dans ce cas, la suppression des conflits a tout intérêt à se faire le plus tôt possible. Hélas, nous ne savons pas à l'avance sur quel cas nous allons tomber. Il incombe donc aux concepteurs de la méthodologie de faire un choix en fonction de ce qu'ils estiment le plus utile pour une personne chargée de l'intégration.

Nous pouvons noter aussi que le même "problème"[12] se pose pour les noms d'objets qui ont une signification proche (ce qu'on appelle synonyme en grammaire): soit cette ressemblance est "voulue" en ce sens que les objets modélisent la même chose[13], soit qu'elle est accidentelle. Les concepteurs de la méthodologie d'intégration pourront généraliser leurs choix à ce problème.

[11] C'est ce que nous appelons par la suite des objets homonymes: des noms identiques sans pour autant représenter la même réalité.

[12] Ce n'est pas un problème à proprement parler pour le modèle (quel qu'il soit) employé car les noms sont différents.

[13] C'est ce que nous appelons par la suite des objets synonymes: des noms différents qui représentent la même réalité.

3.7.3 Conflit de structures

Le problème de structure se pose quand des objets de schémas différents représentent la même réalité par une modélisation différente (par exemple en Entité/Association, un type d'entités a été employé dans un schéma et un type d'associations dans un autre schéma pour représenter le même concept de la réalité).

Il y a différentes façons de résoudre ce problème:

— soit que nous gardons la structure du schéma intermédiaire qui est le résultat de plusieurs autres intégrations et donc le nouveau schéma que nous intégrons peut perdre de sa généralité lors de son incorporation avec le résultat temporaire: cela correspond à prendre la solution la plus restrictive,

— soit que nous prenons la structure la plus générale et nous modifions en conséquence le nouveau résultat temporaire (cette transformation pour obtenir le résultat général peut en effet impliquer des transformations en chaîne).
Un exemple de ces transformations en chaîne, dans le modèle Entité/Association, c'est lorsque nous devons transformer une association en une entité (concept plus général car autonome: il ne dépend pas d'un quelconque contexte). Dans ce cas, il faut intercaler des associations entre la nouvelle entité (l'association que nous venons de transformer) et chaque entité qui participait auparavant à l'association.

La responsabilité du choix incombe au concepteur de la méthodologie d'intégration ou à la personne chargée de l'intégration dans le cas d'une intégration manuelle[14].

3.7.4 Recouvrement de classes

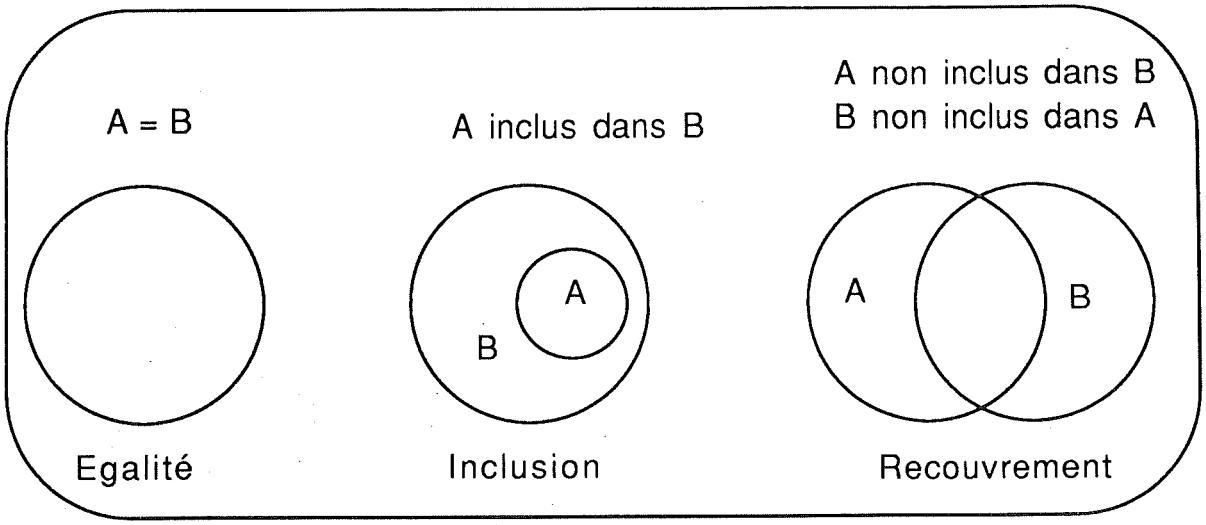
Ce problème, illustré à la Figure 3–12, surgit quand des objets modélisent des concepts du monde réel proches dont des parties se recouvrent. Si cela ne crée pas trop de problèmes pendant le traitement des conflits de noms (voir section 3.7.2), cette situation en pose quand il faut s'occuper des recouvrements de classes. Cela implique de la part des concepteurs de méthodologies d'intégration un choix quant à la façon de transformer les objets pour généraliser la représentation du recouvrement par un seul objet.

Souvent, la solution adoptée est la génération d'un objet sur-type qui englobe les objets d'origine. Cela correspond à ajouter au schéma un objet qui modélisera un concept (à trouver par la personne chargée de l'intégration) plus large qui sera relié aux objets d'origine par des relations spéciales. C'est ce mécanisme qu'on appelle sur-typage (si on le prend du point de vue de l'objet général) ou sous-typage (si on le prend du point de vue des objets d'origine).

La solution du sur-typage n'est valable que pour des modèles très riches de représentation des objets (e.g. le modèle Entité/Association étendu). Si nous voulons appliquer cette solution au modèle relationnel, il faut le faire au moyen de nouvelles relations qui dupliquent l'information de façon adéquate.
Cette duplication peut être trop coûteuse et obliger les concepteurs à chercher des méthodes plus complexes.

[14] Dans ce cas, l'intégrateur est un concepteur de méthodologie puisqu'il applique une certaine démarche pour réaliser l'intégration dont il a la charge.

Figure 3–12: Les différents recouvrements possibles



Exemple 3–3: Ambiguïté du texte

Si les spécifications d'un schéma décrivant la passation de commandes d'un client est définie sans aucune autre information comme suit:

```
"Un client passe des commandes pour des produits . . . ."
```

alors le concepteur ne sait pas dire si on veut modéliser toutes les personnes qui sont effectivement clientes (et avoir un lien obligatoire à la représentation de commande) ou bien si on veut représenter toutes les personnes en se disant qu'elles seront peut-être un jour cliente.

3.7.5 Conflit entre spécifications

Ce problème se pose quand deux schémas décrivent de façon différente la même réalité au moyen de contraintes d'intégrité contradictoires. Un exemple est de dire dans un schéma qu'un objet doit être lié à au moins une relation et dans un autre que le même objet peut ne pas participer à cette relation.
Il ne faut pas confondre ce problème avec celui concernant la structure, car dans un conflit de structures, le problème est au niveau des objets: c'est eux qui sont contradictoires (par exemple, dans le modèle Entité/Association, un concepteur peut avoir représenté un accident par une entité et un autre par une relation entre les entités voitures et propriétaires).

Ce problème peut être dû à une mauvaise perception de la part d'un concepteur de schéma (car il se base sur une description textuelle plus ou moins bien définie mais dont l'interprétation n'est pas toujours univoque).

Ce problème peut être dû à un oubli de la part d'un concepteur de schéma qui n'aura pas vu une clause perdue dans une description trop détaillée.
Ce problème peut enfin être aussi intentionnel de la part d'un concepteur car il se peut que le contexte dans lequel il travaille soit plus strict et dès lors que les contraintes soient plus sévères.

Comme toujours, il appartient à la personne chargée de l'intégration (ou au concepteur de la méthodologie d'intégration) d'opter pour une technique de résolution de ce conflit. Les solutions qui existent (sans être exhaustif) sont:

- prendre les contraintes les plus générales pour couvrir le plus de cas possibles,
- prendre les contraintes les plus sévères pour garantir le maximum de cohérence,
- laisser tomber ce problème et ne pas l'intégrer comme solution de facilité,

.
.
.

tout dépend donc de la politique d'intégration qu'on se choisit, des hypothèses qu'on présuppose sur les schémas qu'on reçoit à intégrer, ...

3.7.6 Restructuration et Enrichissement

Ces problèmes se posent après la phase d'intégration proprement dite. Les techniques de fusion d'objets sont générales pour couvrir le plus de cas possibles mais cela implique que le travail réalisé n'est pas optimal.
Il faut examiner le résultat obtenu après ces transformations pour voir s'il n'y a pas moyen d'optimiser le schéma obtenu afin d'obtenir un schéma plus concis et plus efficace. Il faut cependant être prudent dans cette étude pour restructurer et agir en accord avec les concepteurs.

Une autre raison de cette analyse est qu'elle permet de découvrir de nouvelles propriétés qui n'apparaissaient pas dans les schémas d'origine pris individuellement mais qui voient le jour dans le résultat fusionné.
Un autre usage des enrichissements est fait chez *[Elmasri & Wiederhold 79]*: les auteurs les emploient pour pouvoir intégrer des objets de schémas différents: ils ajoutent des tables (car le modèle qu'ils ont choisi est le modèle relationnel) avec les propriétés communes et les tables d'origine ne contiennent plus que les attributs propres à chaque objet.

Cette phase n'est pas réalisée dans toute méthodologie d'intégration, mais si elle est reprise, il faut que la personne qui accomplit l'intégration en suivant la méthodologie demande aux concepteurs des différents schémas qu'on intègre pour compléter l'information qu'il a à sa disposition afin de pouvoir réaliser cette étape dans les meilleures conditions (et en respectant les souhaits des concepteurs).

3.7.7 Taille des schémas

Ce problème se pose quand les schémas à intégrer sont trop importants ou trop nombreux: dans le cadre d'une démarche manuelle, cela se comprend car la personne chargée de l'intégration peut être submergée par la masse d'informations à analyser (ce qui risque d'introduire des erreurs et des oublis).

Dans les autres démarches (i.e. interactive ou automatique), le problème est aussi présent. La taille des schémas à intégrer implique une explosion combinatoire de comparaisons à effectuer et ce, quel que soit le principe de la méthode d'intégration utilisée.

Le problème se résout facilement s'il est provoqué par le trop grand nombre de schémas à traiter: il suffit de réaliser l'intégration en plusieurs étapes (d'où problème du choix de l'ordre d'intégration).

Si par contre, le problème est dû à de trop gros schémas, il faut voir quels sacrifices l'utilisateur est prêt à concéder.

Dans le cas d'une démarche interactive, il faut bien souvent se résoudre à utiliser un outil qui sacrifie la performance pour préférer un recours minimum à l'utilisateur. La solution idéale serait bien entendu de pouvoir accorder la même attention aux deux critères (i.e. performance et minimum d'interaction).
Pour une méthodologie automatique, une solution pour résoudre ce problème de taille sans trop perdre en performance est l'utilisation de techniques spécialisées dans le traitement de quantités importantes de données.

CHAPITRE 4

COMPARAISON ET CRITIQUE DES DEMARCHES

4.1 Introduction

Après la présentation des principaux concepts de l'intégration (Chapitre 3), ce chapitre est consacré à la comparaison des neuf démarches d'intégration décrites dans le Chapitre 2, ETAT DE L'ART SUR L'INTEGRATION DES VUES, certaines récentes (90) et d'autres plus anciennes (79). Nous utiliserons comme critères: le modèle de données utilisé, le moment où l'intégration prend place dans la démarche de conception de bases de données, les données requises en entrée et en sortie, les différentes étapes que couvre la méthodologie et enfin la stratégie utilisée.

Au cours de ce chapitre, nous mentionnerons parfois des références à des auteurs qui n'ont pas fait l'objet de notre étude dans le Chapitre 2. Ces références serviront à illustrer des cas qui ne font pas partie des méthodologies que nous avons décrites et auxquelles le lecteur intéressé pourra se reporter.

4.2 Les écoles d'intégration

Comme déjà indiqué précédemment, nous recensons quatre écoles:

- l'intégration dynamique,
- l'intégration virtuelle partielle,
- l'intégration virtuelle globale, et
- l'intégration physique globale.

Aucun des auteurs comparés ne fait partie de l'école intégration dynamique, qui intègre au moyen d'un langage de requêtes des bases de données étendu *[Litwin 84]*.

On peut constater deux tendances: l'une reprenant les démarches d'intégration sur les bases de données existantes, ces démarches appartiennent aux écoles d'intégration virtuelle, l'autre comprenant les démarches d'intégration sur les vues, celles-ci appartiennent à l'école d'intégration physique globale.

Tableau 4–1: Ecole d'Intégration

Ecole	Auteurs
intégration virtuelle partielle	*[Motro 87]*
intégration virtuelle globale	*[Dayal & Hwang 84]*
intégration physique globale	*[Elmasri & Wiederhold 79],[Yao & Al 82], [Batini & Lenzerini 84], [Biskup & Convent 86], [Larson & Al 89], [Bouzeghoub & Al 90], [Spaccapietra & Parent 90]*

4.3 Le modèle de données

Nous observons, pour le choix du modèle à utiliser, que les méthodologies les plus anciennes sont basées sur un modèle relationnel (*[Codd 71]*). C'était à l'époque celui qui était le plus développé et le plus utilisé dans la conception de bases de données.

Puis suivant l'approche choisie, les auteurs se sont orientés vers une modélisation plus riche sémantiquement telle que le modèle fonctionnel (pour ceux qui faisaient de l'intégration de bases de données existantes et avaient à comparer des formalismes différents) ou le modèle Entité/Associations (*[Chen 76]*) rapidement étendu par chaque auteur vers un modèle plus puissant (pour ceux qui faisaient de l'intégration de vues).

Les méthodologies les plus récentes *[Spaccapietra & Parent 90]*, *[Bouzeghoub & Al 90]* essaient de combiner les avantages des deux modèles (i.e. fonctionnel et Entité/Association).

Tableau 4–2: Modèles de données utilisés

Auteurs	Modèle utilisé
[Elmasri & Wiederhold 79]	modèle structurel (i.e. relationnel étendu pour modéliser des sous-relations et des associations N-N)
[Yao & Al 82]	modèle fonctionnel FDM et le modèle TASL pour les transactions
[Batini & Lenzerini 84]	modèle Entité/Associations étendu aux concepts d'attributs multivalués, de hiérarchie de classes d'objets et au mécanisme de généralisation/spécialisation
[Dayal & Hwang 84]	modèle fonctionnel FDM étendu au concept de généralisation d'entités et de fonctions
[Biskup & Convent 86]	modèle relationnel, car les auteurs veulent formaliser une théorie d'intégration (étayée par des démonstrations et ont besoin d'un modèle fortement mathématique avec une sémantique stricte)
[Motro 87]	modèle fonctionnel pour représenter les relations ATTribut et GENéralisation
[Larson & Al 89]	modèle ECR (i.e. Entité/Association étendu au concept de catégorie)
[Bouzeghoub & Al 90]	modèle Entité/Association traduit en un réseau sémantique MORSE pour pouvoir être unifié un peu à la manière de PROLOG
[Spaccapietra & Parent 90]	modèle ERC+ (i.e. Entité/Association étendu aux concepts d'attributs multivalués, de généralisation et de relation "may_be_a")

4.4 Position de l'intégration par rapport à la conception

La démarche d'intégration peut avoir lieu à différents moments; tout dépend de la politique adoptée et des outils à sa disposition.

Elle peut avoir lieu:

- lors de l'analyse des besoins *[Khan 79]*,
- lors de la modélisation conceptuelle: pour détecter le plus rapidement possible les conflits inter-schémas et ne pas avoir à répercuter les modifications trop loin *[Teory & Fry 82], [Elmasri & Wiederhold 79], [Batini & Lenzerini 84], [Biskup & Convent 86], [Larson & Al 89], [Bouzeghoub & Al 90], [Spaccapietra & Parent 90]*,
- lors de l'implémentation physique: pour ceux qui étudient aussi les modifications des instructions des programmes de manipulation des données (et ne se contentent pas seulement de mapping vers les données) *[Casanova & Vidal 83], [Yao & Al 82]*, ou
- lors de l'utilisation des bases: pour ceux qui intègrent des bases de données existantes. Dans ce cas, seul le mapping est applicable si on ne veut pas perdre les applications déjà en utilisation *[Dayal & Hwang 84], [Motro 87]*[1].

Tableau 4–3: Position du processus d'intégration

Position	Auteurs
Pendant la phase de conception	*[Elmasri & Wiederhold 79], [Batini & Lenzerini 84], [Biskup & Convent 86], [Larson & Al 89], [Bouzeghoub & Al 90], [Spaccapietra & Parent 90]*
Pendant la phase d'implémentation	*[Yao & Al 82]*
Pendant l'utilisation des BD existantes	*[Dayal & Hwang 84], [Motro 87], ([Spaccapietra & Parent 90])*

Nous remarquons dans le Tableau 4–1 et le Tableau 4–3, que les auteurs se situant dans l'école d'intégration physique globale positionnent le processus d'intégration au moment de l'analyse conceptuelle, alors que les auteurs se situant dans l'école d'intégration virtuelle (partielle ou globale) déclenchent le processus d'intégration au moment de l'utilisation des bases de données.

4.5 Objets en entrée et en sortie

Suivant la stratégie et les étapes que couvrent la méthodologie, les données en entrée doivent plus ou moins être étoffées.

Nous verrons dans le Tableau 4–4, Entrées/Sorties, que les méthodologies ont parfois besoins d'assertions (i.e. contraintes inter-schémas), pour associer des objets qui représentent partiellement ou totalement la même réalité.

[1] La démarche proposée par *[Spaccapietra & Parent 90]* peut être aussi implémentée sur des bases de données existantes.

Tableau 4–4: Entrées/Sorties

Auteurs	Entrées	Sortie
[Elmasri & Wiederhold 79]	deux schémas	un schéma global
[Yao & Al 82]	des vues et les spécifications des requêtes	un schéma global et des requêtes modifiées
[Batini & Lenzerini 84]	des schémas utilisateurs, les poids associés et le schéma de départ	un schéma global
[Dayal & Hwang 84]	des schémas locaux de bases de données existantes et des requêtes	une interface globale vers les bases de données et les requêtes modifiées
[Biskup & Convent 86]	des schémas utilisateurs et les assertions (contraintes d'intégration)	un schéma global sans plus aucune assertion
[Motro 87]	des schémas locaux et leurs requêtes	une "superview" qui sert d'interface entre les diverses BD existantes et les requêtes modifiées
[Larson & Al 89]	deux schémas utilisateurs et le mapping structurel sur les attributs	un schéma global
[Bouzeghoub & Al 90]	des schémas utilisateurs	un schéma global unifié
[Spaccapietra & Parent 90]	deux vues utilisateurs et des assertions concernant les équivalences entre tous les objets	un schéma global et un mapping vers les vues d'origine inchangées

4.6 Etapes de l'intégration couvertes

Les étapes que nous définissons sont:

1. la préintégration,
2. la comparaison et recherche des conflits (mapping structurel),
3. la résolution des conflits de terminologie, et
4. l'intégration découpée en phase de fusion (4a) et restructuration (4b).

La convention utilisée dans le Tableau 4–5, Etapes d'intégration pour indiquer si la démarche reprend une étape sera:

- un tiret "-" si l'étape n'est pas envisagée par la démarche;
- un "X" si l'étape envisagée est en fin de processus;
- un chiffre (qui indique quelle est l'étape suivante) autrement. Nous pouvons en avoir plusieurs (en fait deux), cela signifie qu'il y a deux directions possibles (notamment une boucle de retour en arrière).

Il est possible de classifier les démarches en quatre groupes:

1. celles qui appliquent plusieurs fois les étapes de comparaison, résolution des conflits de terminologie et fusion: *[Elmasri & Wiederhold 79]*;
2. celles qui se concentrent uniquement sur la fusion et restructuration et ignorent les étapes précédentes (qui constituent un problème en elles-mêmes): *[Yao & Al 82]*, *[Biskup & Convent 86]*, *[Motro 87]*, *[Larson & Al 89]*, *[Spaccapietra & Parent 90]*;
3. celles qui font toutes les étapes de l'intégration: *[Dayal & Hwang 84]*, *[Batini & Lenzerini 84]*, *[Bouzeghoub & Al 90]*;

4. celles qui font explicitement mention de la préintégration, notamment de la traduction des schémas en une structure interne *[Bouzeghoub & Al 90]*.

Tableau 4–5: Etapes d'Intégration

Auteurs	étape 1	étape 2	étape 3	étape 4a	étape 4b
[Elmasri & Wiederhold 79]	-	3	4a	2	-
[Yao & Al 82]	-	-	-	4b	4a
[Batini & Lenzerini 84]	-	3	2 et 4a	4b	X
[Dayal & Hwang 84]	-	3	4a	4b	4a
[Biskup & Convent 86]	-	-	-	4b	4b
[Motro 87]	-	-	-	4b	4a
[Larson & Al 89]	-	-	-	4b	X
[Bouzeghoub & Al 90]	2	3	4a	2 et 4b	X
[Spaccapietra & Parent 90]	-	-	-	4b	X

4.7 Stratégie d'intégration

Si presque toutes les méthodologies ne font pas mention explicitement d'une phase de préintégration, elles en ont toutes une, car la décision de la stratégie à adopter pour réaliser la fusion de schémas fait partie de la phase "préintégration".

Il y a deux stratégies d'intégration (section 3.3):

- l'une binaire, qui permet d'intégrer deux schémas à la fois dans un schéma intermédiaire. Elle peut être envisagée de deux façons:
  - — soit l'intégration est équilibrée: le schéma est divisé en paires dès le départ et l'intégration se fait symétriquement *[Teory & Fry 82]*, *[Casanova & Vidal 83]*,
  - — soit l'intégration se fait en échelle: on reprend le résultat intermédiaire et on le fusionne avec un nouveau schéma *[Elmasri & Wiederhold 79]*, *[Batini & Lenzerini 84]*. Cette méthode est un cas particulier de la précédente car si le nouveau schéma qu'on intègre au résultat temporaire n'est pas primitif (i.e. déjà le résultat intermédiaire d'une intégration) on retombe sur la méthode précédente.

  Il est à noter que peu d'auteurs indiquent laquelle des subdivisions ils appliquent. Une des raisons, notamment pour les méthodologies les plus récentes *[Spaccapietra & Parent 90]*, *[Larson & Al 89]*, est que maintenant on se concentre uniquement sur le principe de fusion et les entrées des algorithmes sont constituées de deux schémas.C'est cet algorithme qui se place dans une méthodologie plus vaste (non encore étudiée) où se pose le problème de savoir quels schémas prendre.
- l'autre n-aire (avec n plus grand que 2). Elle peut aussi être envisagée de deux façons:
  - — soit l'intégration en une seule étape: on prend tous les schémas fournis en entrées et on produit un schéma global *[Yao & Al 82]*,
  - — soit l'intégration itérative: on prend plusieurs vues mais pas toutes pour produire un schéma intermédiaire puis ce schéma est repris avec d'autres vues jusqu'à ce qu'il ne reste plus de vues à intégrer *[Navathe & Gadgil 82]*.

Tableau 4–6: Stratégie d'Intégration

Auteurs	Type de stratégie d'intégration	Précision
[Elmasri & Wiederhold 79]	binaire	en échelle
[Yao & Al 82]	n-aire	en une fois
[Batini & Lenzerini 84]	binaire	en échelle
[Dayal & Hwang 84]	binaire	pas de précision
[Biskup & Convent 86]	n-aire	en une fois puis élimination binaire des contraintes d'intégration
[Motro 87]	binaire	pas de précision
[Larson & Al 89]	binaire	pas de précision
[Bouzeghoub & Al 90]	binaire	pas de précision
[Spaccapietra & Parent 90]	binaire	pas de précision

4.8 Détection de conflits

L'activité fondamentale de cette étape consiste à détecter les représentations des mêmes objets du monde réel dans les différents schémas.

Il y a deux types de conflits principalement:

1. les conflits de terminologie (homonymes et synonymes). Les détecter peut donner une indication quant aux relations qui existent entre les représentations des différents schémas à intégrer, et
2. les conflits de structures. Ils arrivent quand les représentations du même objet du monde réel diffèrent par leur structure (Entité vs Attribut par exemple).

Il existe quatre genres de conflits:

- des conflits de types (Entité vs Attribut, . . .),
- des conflits de cardinalités: un lien (d'attribut ou de relation) n'a pas les mêmes minimum et maximum d'occurrence dans les deux schémas (1:1 vs m:n par exemple),
- des conflits de clés, et
- des conflits d'unités de mesure.

Tableau 4–7: Détection des conflits

Auteurs	Conflits de noms	Conflits de structures
[Elmasri & Wiederhold 79]	pas pris en compte	conflits de cardinalité
[Yao & Al 82]	pas pris en compte	pas pris en compte
[Batini & Lenzerini 84]	homonymes synonymes	conflits de types conflits de contraintes d'intégrité

Tableau 4–7 (suite): Détection des conflits

Auteurs	Conflits de noms	Conflits de structures
[Dayal & Hwang 84]	homonymes synonymes	conflits de niveau schéma différence d'échelles différence de structures différence d'abstractions conflits de niveaux données
[Biskup & Convent 86]	pas pris en compte	pas pris en compte
[Motro 87]	pas pris en compte	pas pris en compte
[Larson & Al 89]	pas pris en compte	pas pris en compte
[Bouzeghoub & Al 90]	homonymes synonymes	conflits de structures conflits de contraintes conflits de populations
[Spaccapietra & Parent 90]	pas pris en compte	pas pris en compte

4.9 Technique de fusion choisie

Pour faire la fusion, seul *[Batini & Lenzerini 84]* font un préliminaire en résolvant les conflits de terminologie par renommage et en transformant les objets pour obtenir un schéma où tous les composants sont comparables pour n'avoir plus qu'à faire une superposition des schémas à intégrer. Les autres démarches résolvent les conflits de terminologie et de structures lors de la phase de fusion des vues.

Tableau 4–8: Fusion

Auteurs	Technique d'intégration
[Elmasri & Wiederhold 79]	création de sous-types
[Yao & Al 82]	suppression des fonctions redondantes
[Batini & Lenzerini 84]	en préliminaire élimination des conflits (avec première modification des structures des schémas) puis superposition des schémas: soit on enrichit les objets, soit on crée une généralisation commune ou des sous-types
[Dayal & Hwang 84]	génération d'une interface qui est la généralisation des schémas existants par inclusion, sous-typage, sur-typages, renommage, ou unification des échelles
[Biskup & Convent 86]	regroupement des schémas en un seul puis élimination une par une des contraintes d'intégrations qu'il porte (en transformant le schéma) jusqu'à ne plus en avoir: on a alors un schéma intégré
[Motro 87]	l'utilisateur se génère une interface qui répond juste à ses besoins grâce à un outil qui interprète ses transformations et adapte les requêtes
[Larson & Al 89]	sur base des correspondances entre attributs, l'outil déduit les correspondances entre entités et associations, puis transforme le schéma suivant les correspondances existantes
[Bouzeghoub & Al 90]	les vues sont comparées avec un outil d'unification sémantique et celles qui sont semblables sont transformées pour couvrir la sémantique des schémas d'origine
[Spaccapietra & Parent 90]	après introduction des correspondances par l'utilisateur, l'outil transforme les schémas d'origine en un schéma qui reprend leur sémantique

4.10 Opérations après fusion

Il y a moyen d'optimiser le résultat obtenu, de générer les correspondances entre le schéma intégré et les schémas d'origine (mapping opérationnel), de générer des contraintes (dépendances fonctionnelles, . . .) qui existent entre les objets qu'on vient d'intégrer, d'éliminer la redondance introduite dans les chemins par les opérations de fusion des vues, . . .

Tableau 4–9: Restructuration du schéma intégré

Auteurs	Type d'amélioration
[Elmasri & Wiederhold 79]	-
[Yao & Al 82]	transformer les requêtes pour qu'elles tiennent compte des mappings qui existent
[Batini & Lenzerini 84]	générer les contraintes qui apparaissent dans le schéma intégré, enlever la redondance sur les chemins et normaliser le schéma obtenu
[Dayal & Hwang 84]	générer un mapping qui est appliqué aux requêtes pour qu'elles soient effectives
[Biskup & Convent 86]	générer le mapping de la vue intégrée vers les vues d'origines
[Motro 87]	générer un mapping qui est appliqué aux requêtes pour qu'elles soient effectives
[Larson & Al 89]	-
[Bouzeghoub & Al 90]	générer le mapping de la vue intégrée vers les vues d'origines
[Spaccapietra & Parent 90]	générer le mapping de la vue intégrée vers les vues d'origines

4.11 Nature du processus d'intégration

Tout processus peut être classifié en fonction de son rapport avec l'utilisateur (voir section 3.6) en terme de

- manuel,
- assisté (interactif), ou
- automatique.

Il faut remarquer qu'un processus d'intégration ne peut être entièrement automatique puisque le raisonnement que fait un intégrateur est trop complexe pour être modélisé.
On peut cependant automatiser certaines phases, notamment celle de l'intégration des objets une fois définies toutes les correspondances existantes entre les vues. Pour les autres, on doit garder un certain niveau d'interactivité: tout le problème consiste à trouver le bon compromis entre ce qu'il faut réaliser automatiquement (i.e. le moment où on décide qu'on en sait assez, quitte à se tromper et à obliger l'utilisateur à défaire ce qu'on vient de réaliser), le nombre de questions à poser à l'utilisateur, les suggestions qu'on peut lui faire et ce que l'utilisateur doit réaliser manuellement.

Toutes les méthodologies proposées sont basées au minimum sur l'interactif (sauf une, *[Elmasri & Wiederhold 79]*, qui ne sert qu'à donner une méthodologie formelle à adopter lorsqu'on est confronté à une intégration), mais certaines ne proposent que des indications pour des outils qui supporteraient ces méthodes, d'autres élaborent des techniques plus ou moins poussées.

Les auteurs qui couvrent le moins d'étapes d'intégration se sont souvent concentrés en profondeur sur les parties automatisables *[Spaccapietra & Parent 90], [Larson & Al 89]* et fournissent donc des outils (encore au niveau conceptuel et en développement) plus puissants et entièrement automatiques. Ils ont délibérément supposés comme acquis les phases sujettes à une forte interaction.

Tableau 4–10: Nature du processus d'Intégration

Nature	Auteurs
manuelle	*[Elmasri & Wiederhold 79]*
interactif	*[Yao & Al 82], [Batini & Lenzerini 84], [Dayal & Hwang 84], [Biskup & Convent 86], [Motro 87], [Bouzeghoub 84]*
automatique	*[Larson & Al 89], [Spaccapietra & Parent 90]*

4.12 Bilan et Conclusions des démarches actuelles

La première chose qui apparaît à la lecture de ces résultats est qu'aucune démarche ne propose de solution complète:

soit les auteurs proposent une "solution générale" qui ne fait qu'esquisser des débuts de solutions et ignorent complètement le noeud du problème,
soit les auteurs proposent une solution partielle mais bien étudiée et les étapes d'intégration proposées ne se résument plus à "il n'y a qu'à . . . ".

La tendance est à une spécialisation des problèmes: on va en profondeur quitte à ne pas tout faire.

Une autre constatation qui ressort de l'étude est que les auteurs, au fur et à mesure des découvertes et des articles parus, ont pris conscience de l'importance de laisser les vues d'origine inchangées. Ce qui au départ était une conséquence de la philosophie des écoles d'intégrations virtuelles due au fait que leur objectif était d'intégrer des bases de données existantes, est repris progressivement par les autres écoles pour devenir un "standard" de l'intégration.

Une évidence qui s'impose est que le processus d'intégration est trop complexe et offre une combinatoire de possibilités trop grande que pour être résolu automatiquement (problème indécidable) dans des délais raisonnables. C'est ce qui a poussé les auteurs à peu à peu découper le processus d'intégration en différentes phases (détection des conflits puis fusion) et à se concentrer sur un seul problème à la fois (celui qui est automatisable, à savoir la fusion de schémas sur base de correspondances une fois que celles-ci sont bien (et définitivement) établies).

Ce qui est important aussi, c'est d'avoir un modèle riche en sémantique. S'il a comme inconvénient d'offrir de nombreuses possibilités de représenter différemment le même monde réel, il n'en demeure pas moins un atout pour représenter facilement les objets et faciliter les transformations.

Une solution doit être suffisamment étudiée et formalisée pour offrir un cadre formel à la conception d'outils d'aide à la décision. Les outils, mêmes s'ils sont impossibles à automatiser entièrement, sont d'une grande importance pour aider un administrateur de bases de données à gérer un nombre de plus en plus important de schémas locaux. Un fondement

théorique assez étoffé permet de modulariser en entités autonomes les outils et de diminuer au maximum leur degré d'interactivité. Ces outils peuvent être ainsi regroupés dans un atelier logiciel et enrichir celui-ci pour qu'il incorpore aussi la notion d'intégration de bases de données.

Il reste encore beaucoup à faire dans la première partie du problème de l'intégration (détection des correspondances et/ou des conflits). Quoique certains auteurs proposent une méthode de détection, elle est limitée à la comparaison d'objets de même type (comparaison de deux types d'entités, de deux types d'associations ou de deux attributs *[Bouzeghoub & Al 90]*), si pas carrément superficielle (*[Batini & Lenzerini 84]*).

CHAPITRE 5

METHODOLOGIE GENERALE D'INTEGRATION POUR UN ATELIER LOGICIEL

5.1 Introduction

Dans le Chapitre 4, nous avons comparé un sous-ensemble de méthodologies d'intégration de vues. Dans ce Chapitre 5, nous présentons une méthodologie générale d'intégration pour un environnement d'atelier logiciel.

Nous commençons par définir ce que nous entendons par **atelier logiciel** et les objectifs de celui-ci dans la construction de la base de données d'un système d'information (section 5.2). Dans la section 5.3, nous exposerons le contexte de travail dans lequel la méthodologie a été construite.

Après avoir bien délimité le cadre de travail, nous exposerons la méthodologie (section 5.4), en expliquant la stratégie d'intégration, les typologies de correspondance, le mapping structurel, ainsi que les fonctions générales d'intégration.

Enfin nous clôturerons ce Chapitre 5 en énumérant les fonctions nécessaires qu'un atelier logiciel devrait posséder pour pouvoir appliquer d'un point vue théorique la méthodologie définie. Dans la section 5.6, nous donnerons un algorithme conceptuel implémentant cette méthodologie et nous l'illustrerons par quelques exemples. De l'ensemble des fonctions nécessaires, nous extrairons un noyau minimal de la méthodologie qu'un atelier logiciel doit impérativement avoir pour la réalisation d'un outil d'intégration de vues.

5.2 Définition et Objectifs d'un atelier logiciel

Un atelier logiciel est un ensemble d'outils, aidant les concepteurs à l'élaboration des descriptions des données d'un système d'information. Ces descriptions de données sont effectuées lors de l'analyse conceptuelle à différents niveaux:

- au niveau **CONCEPTUEL ou FONCTIONNEL**, dont l'objectif est d'être une représentation correcte et naturelle du système réel;
- au niveau de **l'IMPLEMENTATION LOGIQUE**, enrichissement des spécifications conceptuelles par la spécification des chemins d'accès aux données nécessaires aux applications.
- au niveau de **l'IMPLEMENTATION PHYSIQUE**, spécification des paramètres physiques, des techniques de stockage et des techniques d'accès.

"Les objectifs de cette hiérarchie sont une réduction de la complexité et l'indépendance des spécifications par rapport à des critères de conception plus techniques" [Hainaut 86a] P17.

Les principales raisons qui ont amené récemment la conception d'ateliers logiciels sont:

- de plus en plus d'intervenants de profils variés participent à la conception. Il faut donc réaliser des outils simples, de très bonne ergonomie, apportant une valeur ajoutée directe à ces nouveaux concepteurs,
- le fait que les tâches de conception se ressemblent d'un système à l'autre: dès lors il est intéressant de fournir une fois pour toute un cadre de conception à la fois correct et réutilisable,
- la conception d'une base de données (complète, cohérente et performante) est la base de la réussite du système d'information. L'atelier doit offrir des outils pour la construction de telles bases.

Nous pouvons observer la valeur ajoutée qu'apporte un atelier logiciel durant l'analyse conceptuelle, depuis la construction du schéma conceptuel jusqu'à la réalisation du schéma physique conforme à un SGBD existant (exemples: Relationnel, Codasyl, . . .)

Les principaux outils qu'offre un atelier logiciel sont des outils nécessaires à l'obtention d'une base de données complète, cohérente et performante dans l'exécution des requêtes sur la base de données. Parmi ceux-ci, nous pouvons citer:

— des outils de construction du schéma de la base de données,

— des outils de validation du schéma de la base de données,

— des outils de transformation d'une structure de données vers une nouvelle structure (par exemple, la transformation d'un attribut en une entité),

— etc, . . .

5.3 Environnement de travail

Comme pour toute méthodologie d'intégration, nous devons définir le cadre de travail dans lequel nous avons construit notre démarche d'intégration. Pour cela, il faut spécifier le modèle de données que nous utilisons, l'école d'intégration à laquelle la méthodologie appartient, le type de démarche d'intégration, ainsi que la nature du processus d'intégration.

Le modèle de données

La plupart des démarches d'intégration exigent que les schémas locaux soient formalisés dans le modèle de données de la démarche. Le choix d'un modèle de données de type relationnel *[Codd 71]* n'est pas un des plus judicieux que nous puissions réaliser, car ce modèle est pauvre dans la représentation sémantique des objets de l'univers du discours et, de plus n'est pas très maniable dans un contexte d'atelier logiciel. Nous nous sommes orientés vers un modèle plus riche syntaxiquement et sémantiquement. Ce modèle peut être un modèle de type Entité/Association *[Chen 76]* ou de type fonctionnel. Nous utiliserons un modèle de données Entité/Association, par le fait même que celui-ci est de plus en plus utilisé dans les ateliers logiciels, mais aussi que c'est ce modèle de données qui est utilisé dans l'atelier logiciel TRAMIS/MASTER, dans lequel nous avons implémenté un petit module de notre méthodologie.

Ce modèle contient les concepts de base suivants:

1. des types d'entités formés d'un ensemble d'attributs,
2. des types d'associations (éventuellement cycliques) entre des types d'entités avec des attributs éventuels,

3. des noms de rôles pour chaque participation d'un type d'entités à un type d'associations. Ces rôles ont une connectivité minimale (min-card) et maximale (max-card),
4. les attributs peuvent être obligatoire (min-card $\geq$ 1) ou facultatifs (min-card = 0), monovaluées (max-card = 1) ou multivaluées (max-card > 1), élémentaires ou décomposables,
5. les types d'entités et les types d'associations peuvent avoir un ou plusieurs ensemble(s) d'attributs et/ou de rôles comme identifiant, et
6. les contraintes.

L'école d'intégration

Nous avons pu voir les différentes écoles d'intégration dans la section 3.4. Le choix de l'école est fonction des objectifs de notre démarche ainsi que de l'environnement de travail dans lequel le processus d'intégration sera déclenché. Le choix d'une école d'intégration dynamique a été écarté car il est peu performant et comme nous travaillons au niveau de l'analyse conceptuelle, il y a des choix plus judicieux. L'école d'intégration virtuelle (partielle ou globale) est aussi rejetée car les données ne sont pas encore présentes (on réalise l'intégration sur des vues) et donc on peut se permettent des modifications pour optimiser des schémas (avec l'intégration physique), ce que l'intégration virtuelle interdit.

Nous avons opté pour l'école d'intégration physique globale. La raison de ce choix est que nous obtenons dans cette école un schéma global dans lequel il est plus facile d'insérer des données lors de l'utilisation de la base nouvellement créée. En effet tout système de gestion de bases de données dispose de mécanismes de vérification de la conformité des données au schéma global.
Comme nous obtenons un schéma global, nous pouvons le retravailler avec un minimum de contraintes: il suffit de conserver la sémantique[1]. De plus l'atelier logiciel ne travaille que sur des vues et non sur des bases de données existantes.

La démarche

Nous avons vu à la section 3.5 que le processus d'intégration était subdivisé en plusieurs phases dont les deux plus importantes sont: la phase de mise en correspondance (mapping structurel) et la phase d'intégration. Notre démarche ne s'occupe que de la phase d'intégration (i.e. la résolution des conflits qui ont été exprimés lors de la phase précédente dans l'établissement du mapping structurel entre les schémas à intégrer). La phase de mise en correspondance est définie par le concepteur, lequel établit le mapping structurel manuellement. Ce choix a pour but de simplifier la définition de notre méthodologie, mais l'autre phase pourra faire l'objet d'une recherche future.

La nature d'Intégration

Il est possible d'obtenir trois processus d'intégration de nature différente: manuelle, interactive ou automatique (voir la section3.6). Un processus manuel est envisageable uniquement pour l'intégration de petits schémas locaux. Quant au processus automatique, il a l'avantage d'être efficace par le fait que le concepteur ne doit pas intervenir lors de la phase d'intégration, mais l'inconvénient majeur d'un tel processus est qu'il donne souvent un schéma intégré incompréhensible aux yeux du concepteur et ne représente pas toujours les objets comme le concepteur l'aurait désiré. Pour notre part, nous nous sommes orientés vers un processus de nature interactive. Cela permet de mieux impliquer les concepteurs dans

[1] Puisque nous sommes en phase de conception, les transformations sont plus faciles: elles ne portent que sur les structures de données.

le processus d'intégration et d'obtenir un schéma conforme, complet et compréhensible pour les concepteurs. De plus nous ne cherchons pas à obtenir un processus d'intégration performant dans un premier temps mais bien un processus réalisant les objectifs de l'intégration des vues.

5.4 Méthodologie générale d'intégration

5.4.1 Stratégie d'intégration pour un atelier logiciel

5.4.1.1 Applicabilité

Nous pouvons lire dans la section 5.2 que les ateliers logiciels travaillent au niveau de l'analyse conceptuelle, afin d'offrir une aide aux concepteurs lors de la description des données d'un système d'information. La définition d'un processus d'intégration ne peut se réaliser qu'au moment de la conception des bases, puisque de tels outils se situent dans une école d'intégration physique globale.

Le problème reste de choisir à quel moment il faut déclencher le processus d'intégration des vues (voir Figure 5–1). Il est possible de le faire à trois moments distincts:

— au niveau de l'analyse des besoins,

— au niveau du schéma conceptuel, ou

— au niveau du schéma physique.

L'exécution du processus d'intégration au niveau de l'analyse des besoins rend la définition de la méthodologie difficile, car à ce moment les schémas sont très pauvres en structure. De plus la méthodologie devra être construite sur une analyse sémantique des concepts des schémas à intégrer, ce que nous n'avons pas encore au niveau de l'analyse des besoins.

D'un autre côté, si nous pouvions placer le processus d'intégration au niveau du schéma physique, il y aurait encore certaines difficultés à définir la méthode d'intégration, puisque à ce niveau, les représentations sémantiques de l'univers du discours ne sont plus les meilleures.

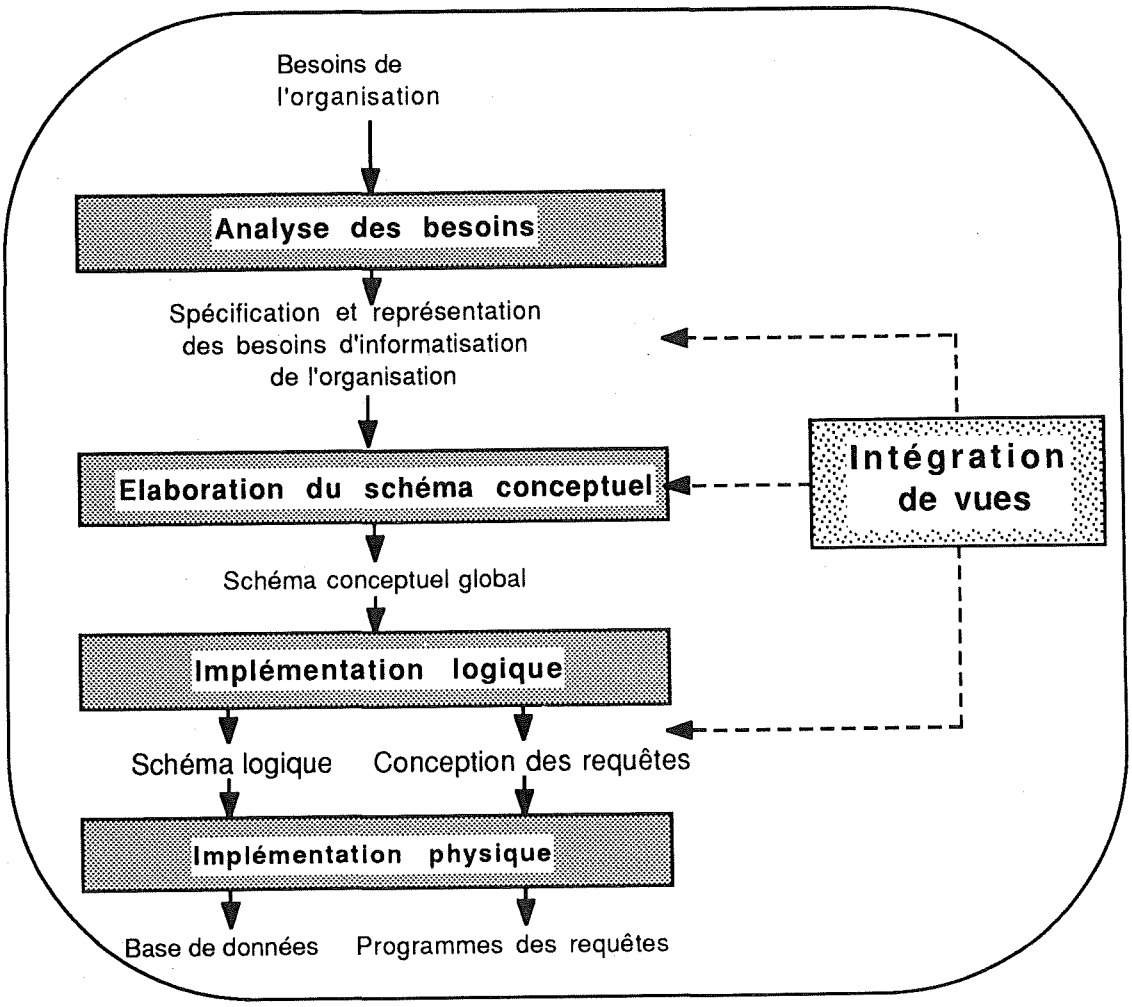
Il reste l'intégration au niveau du schéma conceptuel, qui est le moment idéal pour le déclenchement du processus dans un atelier logiciel. Ce choix est celui de la plupart des démarches d'intégration des vues (voir Tableau 4–3). Il vaut mieux exécuter l'intégration le plus tôt possible, car le coût du traitement d'une erreur augmente dans les différentes étapes du cycle de développement de la base de données et le lancement du processus d'intégration ne peut être réalisé que sur des représentations acceptables (complètes, correctes, minimales et non ambiguës).

5.4.1.2 Intégration binaire

Il existe deux stratégies d'intégration: binaire et n-aire (voir section 3.3). La stratégie n-aire[2] a comme avantage d'obtenir un schéma intégré en déclenchant une seule fois le processus d'intégration. Le désavantage de cette stratégie est que cela rend complexe la définition d'une méthodologie d'intégration n-aire. Par contre la stratégie binaire permet d'avoir une intégration plus simple dans les activités de comparaison et de résolution des

[2] n>2 où n est le nombre de schémas à intégrer.

Figure 5–1: Phases de conception de la base de données



conflits, mais cela est fait au détriment de la performance, par l'exécution répétée du processus d'intégration.

Un des buts actuels du processus d'intégration dans un atelier logiciel, est d'obtenir un schéma intégré reprenant de façon complète, conforme, compréhensible et sans redondance toutes les représentations de l'univers du discours des schémas à intégrer. De plus l'objectif d'avoir une démarche la plus efficace possible passe en second plan. C'est en fonction de quoi nous pouvons dire qu'une stratégie d'intégration binaire est un choix judicieux dans la définition d'une méthode d'intégration pour un atelier logiciel.

Nous pouvons aussi constater que la plupart des démarches d'intégration (voir Tableau 4–6) des vues utilisent cette stratégie, afin de simplifier la définition de la méthodologie.

5.4.1.3 Politique d'intégration

Il est possible de définir deux politiques générales d'intégration binaire dans un atelier logiciel. La première politique consiste à intégrer les deux schémas (S1, S2) en générant un nouveau schéma (S3) où S3 = (S1\S2) + (S2\S1) + (S1 ∩ S2). Pour ce qui est dans l'intersection, on choisit la solution la plus générale. Cette politique permet la conservation des schémas locaux S1,S2.
Dans la deuxième politique, on part du principe d'enrichir un schéma (soit Sp, dit primaire) par un autre (soit Ss, dit secondaire). On obtient après le processus d'intégration Sp = Sp + (Ss\Sp)[3], avec Ss inchangé. Si on veut garder inchangés les schémas locaux comme dans la première politique, il suffit de faire une copie de Sp avant le déclenchement du processus d'intégration.

Nous avons opté pour une variante de la deuxième politique. Cette variante consistera à prendre, en cas de conflit de structures, la structure la plus générale (e.g. le type d'entités).

Il est possible d'appliquer deux variantes à chacune de ces politiques: l'une consiste à avoir un processus d'intégration "OFF-LINE", tandis que l'autre consiste à avoir une intégration "ON-LINE".
Dans la première variante, on déclenche le processus d'intégration une fois le mapping structurel défini entre les deux schémas. Alors que dans la deuxième variante on a un déclenchement du processus d'intégration pour chaque correspondance définie par l'utilisateur.

La politique de notre démarche est l'intégration d'un schéma dans un autre de façon "ON-LINE". Le but essentiel est d'offrir la possibilité d'impliquer le concepteur dans la phase d'intégration, avec la possibilité pour celui-ci de suivre le processus pas à pas et de pouvoir arrêter le processus à n'importe quel moment. L'utilisateur aura aussi la possibilité de reprendre ce processus par après.

En bref, l'utilisateur donnera une à une chaque correspondance entre les deux schémas. Le processus d'intégration résoudra la correspondance et il interagira éventuellement avec l'utilisateur afin que celui-ci donne son accord interactivement.

5.4.1.4 Détection des homonymes

Il est possible d'offrir à l'utilisateur un outil simple de détection des homonymes. Il suffit pour cela de comparer, avant le déclenchement des processus d'intégration, chaque nom des concepts d'un schéma avec les noms des concepts de l'autre schéma.

Il faut bien remarquer que cet outil de détection ne permet que la découverte des homonymes au niveau syntaxique. La détection au niveau sémantique n'est pas prise en considération.

Cet outil génèrera une liste de correspondances comprenant les concepts (identiques) ayant le même nom et représentant le même univers du discours, ainsi que les concepts (contradictoires) ayant le même nom mais d'univers du discours différents.

[3] Cette notation signifie que l'on enrichit la structure de Sp (sans la modifier) par la structure se trouvant dans Ss mais pas dans Sp.

5.4.2 La typologie des correspondances

La typologie de notre méthode est basée sur celle adoptée par *[Spaccapietra & Parent 90]*, c'est-à-dire que les équivalences dont nous parlons seront des équivalences sémantiques. Nous ne regarderons, pour établir une équivalence entre deux concepts de deux schémas à intégrer, que les univers du discours de chacun de ces concepts et pas les structures et propriétés de ces concepts.

Cette optique est plus souple et plus riche pour décrire les correspondances. Elle permet d'imposer moins de préconditions à l'établissement d'une correspondance.

Nous avons besoin pour décrire cette typologie de la définition d'univers du discours (Real World State), que l'on retrouve chez *[Larson & Al 89]*.

L'univers du discours (UDD) est " *l'ensemble des instances du monde réel d'une classe d'objets à un moment. UDD est un concept semblable à l'extension de bases de données sauf qu'il fait référence au monde réel sous-jacent des instances de l'objet représenté. Quand deux extensions distinctes d'objets A et B dans des bases de données différentes représentent en fait les instances des mêmes objets du monde réel, on peut dire que l'extension de A n'est pas égale à l'extension de B. Cependant UDD(A) = UDD(B)" [Larson & Al 89].*

Dans la suite nous parlerons de **parent** quand nous nous intéresserons à un objet (entité, association ou attribut) qui possède l'attribut dont nous sommes en train de parler.

La typologie adoptée tient compte du contexte de travail que nous avons choisi, notamment l'hypothèse disant que les correspondances ont un sens: nous introduisons une correspondance du schéma secondaire (soit Ss) vers le schéma primaire (soit Sp).

Correspondances ENTITE/ENTITE

a. **Equivalence** entre E1 et E2 avec E1 appartenant à Ss et E2 à Sp:

   OU

   — les univers du discours sont égaux (i.e. UDD(E1) = UDD(E2)),

   — il y a une fonction bijective totale entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique.

b. **Inclusion** de E1 dans E2 avec E1 appartenant à Ss et E2 à Sp:

   OU

   — l'univers du discours de E1 est inclus dans celui de E2 (i.e. $UDD(E1) \subseteq UDD(E2)$),

   — il y a une fonction injective totale entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique.

c. **Compréhension** de E1 dans E2 avec E1 appartenant à Ss et E2 à Sp:

   OU

   — l'univers du discours de E1 comprend celui de E2 (i.e. $UDD(E1) \supseteq UDD(E2)$),

   — il y a une fonction bijective partielle entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique.

d. **Recouvrement** de E1 et de E2 avec E1 appartenant à Ss et E2 à Sp:

OU

— l'intersection des univers du discours de E1 et de E2 n'est pas vide (i.e. $UDD(E1) \cap UDD(E2) \neq \emptyset$),

— il y a une fonction injective partielle entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique.

e. **Exclusion** de E1 et de E2 avec E1 appartenant à Ss et E2 à Sp:

OU

— l'intersection des univers du discours de E1 et de E2 est vide (i.e. $UDD(E1) \cap UDD(E2) = \emptyset$),

— il n'y a pas de fonction entre les deux univers du discours qui associerait les objets avec la même sémantique, c'est à dire que les UDD représentent des occurrences différentes (dans un même domaine).

Correspondances ASSOCIATION/ASSOCIATION

Pour que deux associations soient en correspondance, il faut que toutes les entités qui participent à l'association du schéma Ss soient en correspondance avec un objet du schéma Sp dans lequel nous voulons intégrer.
Pour le reste, la même classification que celle adoptée pour les entités peut s'opérer pour les associations:

a. **Equivalence** entre R1 et R2 avec R1 appartenant à Ss et R2 à Sp:

ET

+ OU

— les univers du discours sont égaux (i.e. UDD(R1) = UDD(R2)),

— il y a une fonction bijective totale entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique;

+ toutes les entités qui participent à l'association R1 sont en correspondance d'équivalence avec des objets de Sp.

b. **Inclusion** de R1 dans R2 avec R1 appartenant à Ss et R2 à Sp:

ET

+ OU

— l'univers du discours de R1 est inclus dans celui de R2 (i.e. $UDD(R1) \subseteq UDD(R2)$),

— il y a une fonction injective totale entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique;

+ toutes les entités qui participent à l'association R1 sont en correspondance d'équivalence ou d'inclusion avec des objets de Sp;

+ au moins une des correspondances dans laquelle participe une entité de Ss est une correspondance d'inclusion.

c. **Compréhension** de R1 dans R2 avec R1 appartenant à Ss et R2 à Sp:

ET

+ OU
  - — l'univers du discours de R1 comprend celui de R2 (i.e. $UDD(R1) \supseteq UDD(R2)$),
  - — il y a une fonction bijective partielle entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique;
+ toutes les entités qui participent à l'association R1 sont en correspondance d'équivalence ou de compréhension avec des objets de Sp;
+ au moins une des correspondances impliquant une entité de Ss est une correspondance de compréhension.

d. **Recouvrement** de R1 et de R2 avec R1 appartenant à Ss et R2 à Sp:

ET

+ OU
  - — l'intersection des univers du discours de R1 et de R2 n'est pas vide (i.e. $UDD(R1) \cap UDD(R2) \neq \emptyset$),
  - — il y a une fonction injective partielle entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique.
+ toutes les entités qui participent à l'association R1 sont en correspondance d'équivalence, d'inclusion, de compréhension ou de recouvrement avec des objets de Sp;
+ OU
  - — au moins une des correspondances impliquant une entité de Ss est une correspondance de recouvrement,
  - — ET
    * au moins une des correspondances impliquant une entité de Ss est une correspondance d'inclusion,
    * au moins une des correspondances impliquant une entité de Ss est une correspondance de compréhension.

e. **Exclusion** de R1 et de R2 avec R1 appartenant à Ss et R2 à Sp:

ET

+ OU
  - — l'intersection des univers du discours de R1 et de R2 est vide (i.e. $UDD(R1) \cap UDD(R2) = \emptyset$),
  - — il n'y a pas de fonction entre les deux univers du discours qui associerait les objets avec la même sémantique, c'est-à-dire que les UDD représentent des occurrences différentes (dans un même domaine);
+ au moins une des entités qui participent à l'association R1 est en correspondances d'exclusion avec un objet du schéma Sp.

Correspondances ATTRIBUT/ATTRIBUT

Pour que deux attributs soient en correspondance, il faut que le parent de l'attribut dans Ss soit en correspondance avec un objet (pas forcément de même type) dans Sp.

a. **Equivalence** entre A1 et A2 avec A1 appartenant à Ss et A2 à Sp:

   OU

   — les univers du discours sont égaux (i.e. UDD(A1) = UDD(A2)),

   — il y a une fonction bijective totale entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique.

b. **Inclusion** de A1 dans A2 avec A1 appartenant à Ss et A2 à Sp:

   OU

   — l'univers du discours de A1 est inclus dans celui de A2 (i.e. $UDD(A1) \subseteq UDD(A2)$),

   — il y a une fonction injective totale entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique.

c. **Compréhension** de A1 dans A2 avec A1 appartenant à Ss et A2 à Sp:

   OU

   — l'univers du discours de A1 comprend celui de A2 (i.e. $UDD(A1) \supseteq UDD(A2)$),

   — il y a une fonction bijective partielle entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique.

d. **Recouvrement** de A1 et de A2 avec A1 appartenant à Ss et A2 à Sp:

   OU

   — l'intersection des univers du discours de A1 et de A2 n'est pas vide (i.e. $UDD(A1) \cap UDD(A2) \neq \emptyset$),

   — il y a une fonction injective partielle entre les deux univers du discours, telle que deux objets sont reliés si et seulement si ils ont la même sémantique.

e. **Exclusion** de A1 et de A2 avec A1 appartenant à Ss et A2 à Sp:

   OU

   — l'intersection des univers du discours de A1 et de A2 est vide (i.e. $UDD(A1) \cap UDD(A2) = \emptyset$),

   — il n'y a pas de fonction entre les deux univers du discours qui associerait les objets avec la même sémantique, c'est à dire que les UDD représentent des occurrences différentes (dans un même domaine).

Correspondances mixtes

Lorsque nous avons une correspondance mixte[4], nous obtenons la même classification que pour les correspondances entre les objets de même type, à savoir:

a. équivalence,

b. inclusion,

c. compréhension,

[4] Il faut que les objets mis en correspondance soient de types différents (par exemple, une entité et un attribut).

d. recouvrement, ou

e. exclusion.

Nous établissons ces correspondances mixtes de la même manière en se basant sur la signification des univers du discours respectifs de chacun des objets participant à la correspondance,

Les conditions à remplir pour chacune de ces correspondances sont:

- entre Entité et Attribut,
  les mêmes qu'entre Entité et Entité: un attribut peut être considéré comme une entité "dégénérée", c'est-à-dire que le concepteur a voulu lui accorder une importance moindre ou en tout cas n'a pas estimé ce concept comme autonome. Il existe une transformation qui permet de passer facilement (et sans condition) d'un attribut à une entité;
- entre Entité et Association,
  les mêmes qu'entre Entité et Entité: une association peut être considérée comme un "raccourci", c'est-à-dire que le concepteur a voulu garder le schéma plus compact. Il existe une transformation qui permet de passer facilement (et sans condition) d'une association à une entité;
- entre Association et Entité (ou Attribut),
  les mêmes qu'entre Entité et Entité plus la condition que les entités participant à l'association soient déjà en correspondance avec des objets de l'autre schéma car une association n'a de sens que par la participation d'au moins deux entités à celle-ci: il faut donc que ces entités soient mises en correspondance pour garder la même sémantique;
- entre Attribut et Entité (ou Association),
  les mêmes qu'entre Entité et Entité plus la condition que le parent de l'attribut soit déjà en correspondance avec un objet de l'autre schéma car un attribut n'a pas de signification sans son parent auquel il ajoute une propriété: il faut que celui-ci soit mis en correspondance pour garder la même sémantique;
- Nous avons finalement une correspondance mixte spéciale impliquant le vide: c'est une correspondance entre un objet de Ss et un objet qui n'existe pas dans Sp. Cette correspondance permet de créer dans Sp des objets qui sont présents seulement dans Ss.

  La correspondance ainsi établie sera une correspondance d'équivalence entre deux objets de même type puisqu'en copiant l'objet dans le schéma primaire Sp, nous introduisons un objet identique à celui présent dans le schéma secondaire Ss qui vérifie donc les conditions de la correspondance d'équivalence entre deux objets de même type (par exemple, Entité/Entité).

5.4.3 Le Mapping Structurel

Dans cette section nous définissons ce que nous entendons par **Mapping Structurel**. Nous donnons ensuite les hypothèses que nous prenons comme préalable à notre méthodologie d'intégration et la syntaxe (en notation BNF) que nous adoptons pour ce mapping.

Définition

Le **mapping structurel:** *c'est l'ensemble des correspondances établies jusqu'à présent entre les schémas Ss et Sp et qui ont déjà été intégrées dans le schéma primaire Sp.*

Hypothèses

Nous supposerons dans la méthodologie que:

1. le libellé des objets (entités ou associations) est unique au sein du schéma auquel les objets appartiennent,
2. le libellé des attributs est unique au sein de l'objet (entité ou association) auquel les attributs appartiennent: un libellé d'attribut n'est identifiant que localement à l'objet auquel il appartient (et pour un même niveau) et pas globalement à tout le schéma,
3. les deux schémas à intégrer (soient Ss et Sp) ont été correctement formalisés dans le modèle E/A et sont complets et cohérents.

Syntaxe de désignation des correspondances

Cette syntaxe, de même que le principe du mapping structurel, est basée sur le travail de *[Belfar 84]*. Nous utiliserons les symboles suivants:

E: désignera une entité (Entity),
R: désignera une association (Relationship),
A: désignera un attribut (Attribute),et
N: désignera le concept "rien" (Nothing) qui n'est aucun des trois concepts ci-dessus mais qui jouera la même fonction que l'objet avec lequel il est mis en correspondance.

Les correspondances seront exprimées par:

- EE = correspondance Entité/Entité,
- RR = correspondance Association/Association,
- AA = correspondance Attribut/Attribut,
- EA = correspondance Entité/Attribut,
- AE = correspondance Attribut/Entité,
- ER = correspondance Entité/Association,
- RE = correspondance Association/Entité,
- RA = correspondance Association/Attribut,
- AR = correspondance Attribut/Association,
- EN = correspondance Entité/Vide,
- RN = correspondance Association/Vide, et
- AN = correspondance Attribut/Vide.

Nous aurons donc le mapping structurel de Ss vers Sp (noté MS(Ss, Sp)) qui contiendra toutes les correspondances introduites par l'utilisateur depuis le début de l'intégration en cours. Ce mapping sera un objet (de type ensemble) qui vérifiera la syntaxe suivante:

- mapping ::= MS(<nom de schéma Ss>, <nom de schéma Sp>); <liste des correspondances>,
- <liste des correspondances> ::= ∅ | <correspondance>; <liste des correspondances>,
- <correspondance> ::= <Type EE> | <Type RR> | <Type AA> | <Type EA> | <Type AE> | <Type ER> | <Type RE> | <Type RA> | <Type AR> | <Type EN> | <Type RN> | <Type AN>,

- <Type EE> ::= EE <libellé de l'entité du schéma Ss>*<Type de correspondance E>*<libellé de l'entité du schéma Sp>,
- <Type RR> ::= RR <libellé de l'association du schéma Ss>*<Type de correspondance R>*<libellé de l'association du schéma Sp>,
- <Type AA> ::= AA <libellé de l'attribut du schéma Ss>.<libellé de son propriétaire>*<Type de correspondance A>*<libellé de l'attribut du schéma Sp>.<libellé de son propriétaire>,
- <Type EA> ::= EA <libellé de l'entité du schéma Ss>*<Type de correspondance E>*<libellé de l'attribut du schéma Sp>.<libellé de son propriétaire>,
- <Type AE> ::= AE <libellé de l'attribut du schéma Ss>.<libellé de son propriétaire>*<Type de correspondance E>*<libellé de l'entité du schéma Sp>,
- <Type ER> ::= ER <libellé de l'entité du schéma Ss>*<Type de correspondance E>*<libellé de l'association du schéma Sp>,
- <Type RE> ::= RE <libellé de l'association du schéma Ss>*<Type de correspondance E>*<libellé de l'entité du schéma Sp>,
- <Type RA> ::= RA <libellé de l'association du schéma Ss>*<Type de correspondance E>*<libellé de l'attribut du schéma Sp>.<libellé de son propriétaire>,
- <Type AR> ::= AR <libellé de l'attribut du schéma Ss>.<libellé de son propriétaire>*<Type de correspondance E>*<libellé de l'association du schéma Sp>,
- <Type EN> :== EN <libellé de l'entité du schéma Ss>,
- <Type RN> :== RN <libellé de l'association du schéma Ss>,
- <Type AN> :== AN <libellé de l'attribut du schéma Ss>.<libellé de son propriétaire>,
- <Type de correspondance E> ::= Equivalent | Inclusion | Compréhension | Recouvrement | Exclusion,
- <Type de correspondance R> ::= Equivalent | Inclusion | Compréhension | Recouvrement | Exclusion, et
- <Type de correspondance A> ::= Equivalent | Inclusion | Compréhension | Recouvrement | Exclusion.

La signification des symboles adoptés ci-dessus est la suivante:

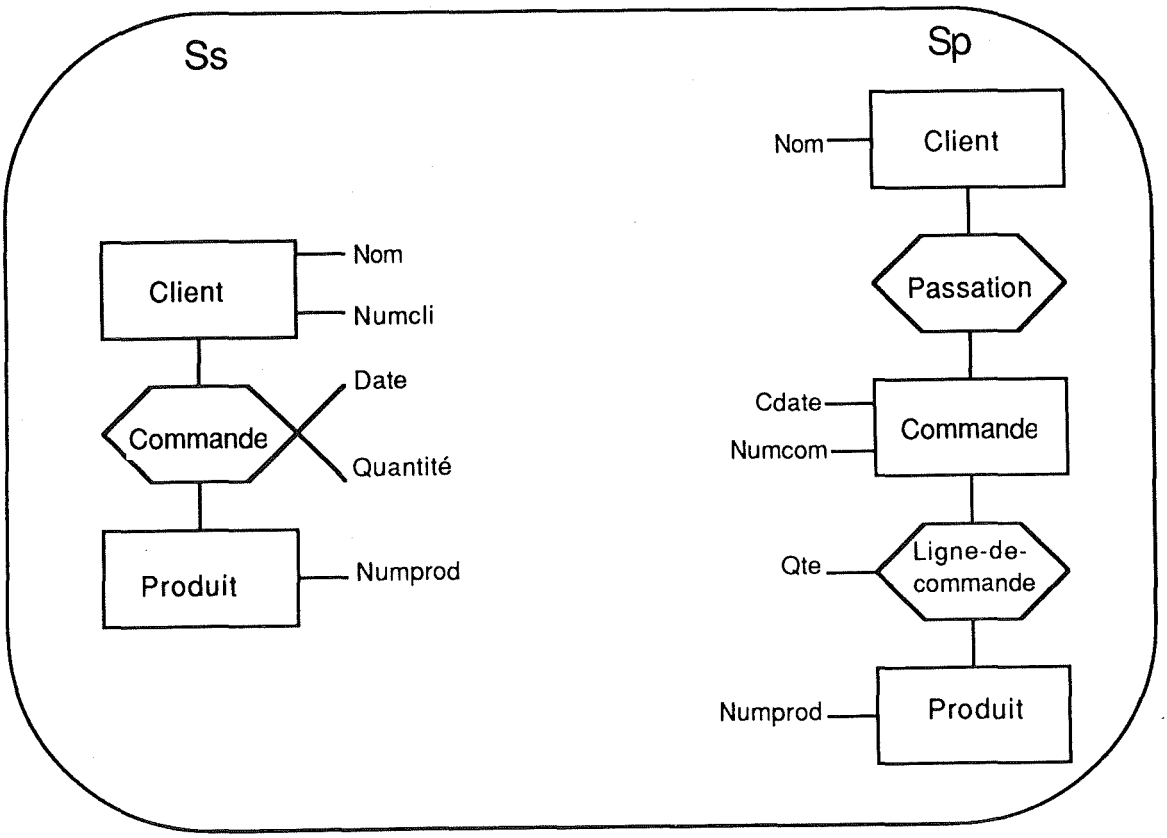
∅ représente l'ensemble vide,
| signifie un ou exclusif,
";", "." et "*" sont des caractères qui serviront dans la syntaxe de séparateurs d'objets.

L'Exemple 5–1 montre le mapping structurel que devrait introduire l'utilisateur pour réaliser l'intégration du schéma secondaire (Ss) dans le schéma primaire (Sp) de la Figure 5–2.

Exemple 5–1: Illustration d'un mapping structurel

```
MS(Ss,Sp) =
          EE Client*Equivalent*Client
           AA Nom.Client*Equivalent*Nom.Client
           AN Numcli.Client

          EE Produit*Equivalent*Produit
           AA Numprod.Produit*Equivalent*Numprod.Produit

          RR Commande*Equivalent*Ligne_de_commande
           AA Quantite.Commande*Equivalent*Qte.Ligne_de_commande
           AA Date.commande*Equivalent*Cdate.Commande
```

Figure 5–2: Les schémas Commande-Client

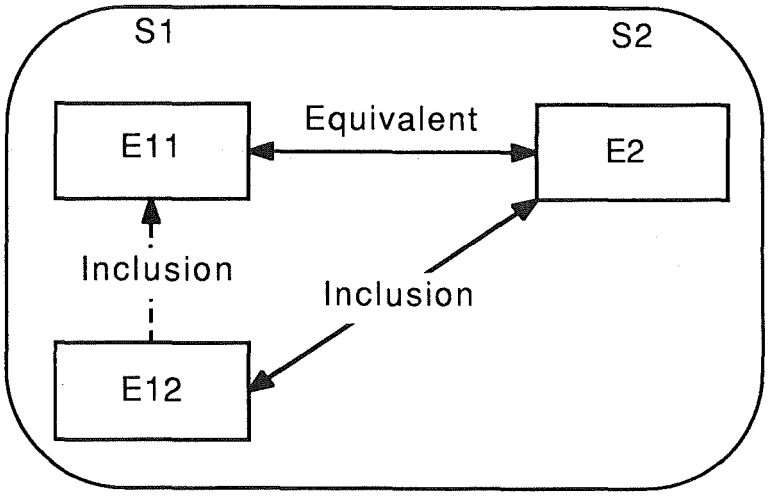


5.4.4 Cohérence du mapping structurel

Nous dirons que le mapping introduit jusqu'à présent par l'utilisateur est correct si chaque objet d'un schéma (entité, association, . . .) ne peut être en correspondance (de n'importe quel type i.e. équivalence, inclusion, . . .) qu'avec un et un seul objet de l'autre schéma.

Cette définition est cohérente avec les hypothèses faites sur les schémas à intégrer (complétude et cohérence) car si nous avions d'une part une correspondance d'équivalence entre les entités E11 et E2 (voir Figure 5–3), et d'autre part une correspondance d'inclusion entre E12 et E2, cela signifierait que nous aurions dû avoir une inclusion de E12 dans E11 (sous-typage) qui n'aurait pas été modélisée, ce qui est contradictoire avec les hypothèses disant que les schémas en entrée sont corrects.

Figure 5–3: Incohérence au niveau du mapping structurel



Remarque

La contrainte d'exclusion n'a de sens que pour indiquer que deux entités ont des domaines comparables mais des occurrences différentes. Cela implique qu'elles appartiennent à un sur-type commun. Dès lors l'introduction d'une contrainte d'exclusion impliquera automatiquement une contrainte d'inclusion avec le parent (sur-type) d'une des deux entités.

Le seul problème qui pourrait se poser à "longue échéance" (i.e. lors de l'intégration d'un troisième schéma), c'est lorsqu'on a des contraintes d'exclusion et que le parent commun n'est pas présent.
Dans ce cas, lors de l'apparition du sur-type parent, il faudrait introduire deux correspondances de compréhension impliquant la même entité (ce qui est interdit). Dès lors pour contourner ce problème, il faudra créer une entité sur-type (si le parent est absent) dès que nous introduisons une contrainte d'exclusion. Lorsqu'apparaîtra l'entité qui joue le rôle de sur-type, elle devra être mise en correspondance par l'utilisateur avec le parent que nous avons introduit anticipativement.

5.5 Les fonctions d'intégration nécessaires

Nous spécifions dans cette section l'ensemble des fonctions d'intégration nécessaires qui peuvent être définies lors de l'intégration d'un schéma (soit Ss, dit schéma secondaire) dans un autre (soit Sp, dit schéma primaire). Nous limitons cette liste au type de correspondance d'équivalence parmi les objets[5] des deux schémas. Tous les autres types de correspondances peuvent se ramener au cas de l'intégration de correspondance d'équivalence.

[5] Nous entendons par objet une entité, une association ou un attribut.

Limitation à l'équivalence

Lors d'une correspondance d'inclusion entre entités, un moyen de s'en sortir est de créer une entité sous-type de l'entité mise en correspondance dans le schéma primaire. Cette nouvelle entité est en correspondance d'équivalence avec l'entité du schéma secondaire.
Une correspondance de compréhension entre entités peut être traitée en créant une entité sur-type de l'entité mise en correspondance dans le schéma primaire. Cette nouvelle entité est en correspondance d'équivalence avec l'entité du schéma secondaire.
Une correspondance de recouvrement entre entités peut être traitée en créant une entité sur-type (soit ST) de l'entité mise en correspondance dans le schéma primaire, puis créer une entité sous-type de ST. Cette dernière entité est en correspondance d'équivalence avec l'entité du schéma secondaire.
Une correspondance d'exclusion entre entités se traite comme le recouvrement sauf qu'il faut ajouter en plus une contrainte d'exclusion mutuelle.

Dès lors, il n'y a plus moyen d'introduire autre chose que des correspondances d'équivalence impliquant une association dans le schéma secondaire.

Pour les attributs, si nous n'avons pas une correspondance d'équivalence, c'est qu'il y a une différence d'agrégation dans les deux schémas. Cette différence est éliminée en désagrégeant la structure la plus compacte.
Pour les correspondances entre attribut et entité, il suffit de "voir" l'attribut comme une entité pour se ramener aux cas énoncés plus haut.
Pour les correspondances entre attribut et association, si l'association ne reprend pas entièrement la sémantique de l'attribut (cas des correspondances de compréhension, de recouvrement et d'exclusion), il faut créer un attribut à l'objet (dans le schéma primaire) en correspondance avec le parent (dans le schéma secondaire).
Quand nous créons un attribut dans le schéma primaire, il est bien évidemment en correspondance d'équivalence avec celui du schéma secondaire.

Termes employés

La description des fonctions d'intégration est faite en terme de préconditions et d'actions (effectuée sur le schéma primaire et sur l'ensemble MS[6] par l'ajoute d'une nouvelle CS[7]). Nous utilisons la notation suivante:

— Ss,Sp : deux schémas conceptuels dont nous voulons intégrer **Ss dans Sp**;

— E: **entité** de libellé E;

— R: **association** de libellé R;

— R (E1[min1,max1], . . . , En[8][minn,maxn]): désigne l'**association** R avec les entités E1, . . . , En (et leurs connectivités respectives) qui participent à l'association R;

— A: **attribut** de libellé A;

— O: **objet** de libellé O;

— A.O: désigne l'attribut A de l'objet O;

— A.O [min-con,max-con] : désigne l'attribut A de l'objet O avec ses **connectivités minimale et maximale**;

---

[6] MS: Mapping Structurel.

[7] CS: une Correspondance structurelle du schéma secondaire vers le schéma primaire.

[8] n: désigne le degré de l'association et celui-ci doit toujours être strictement supérieur à 1.

— lien-A (A,O): désigne le lien entre l'attribut A et l'objet O;

— lien-R (E,R): désigne le lien entre l'entité E et l'association R;

— chemin(O,D) est une succession de liens (lien-A et/ou lien-R) entre l'objet O et l'objet D. Un chemin est dit de longueur 0 si l'origine et la destination représentent le même objet. lien-A et lien-R sont deux chemins de longueur 1;

— CREER-TE(E): définit une **nouvelle entité** de libellé E dans le schéma primaire;

— CREER-TA(R): définit une **nouvelle association** de libellé R dans le schéma primaire;

— CREER_ATT(A.O) définit un **nouvel attribut** de libellé A à l'objet O dans le schéma primaire;

— SUPPRIMER_ATT(A,O) supprime l'attribut A et le lien-A à l'objet O dans le schéma primaire;

— SUPPRIMER_TA(R) supprime l'association R et les liens-R aux entités qui participaient à l'association R dans le schéma primaire.

5.5.1 Transformations d'un objet en une entité

Lors de l'application des différentes fonctions d'intégration, il nous faudra parfois transformer dans le schéma primaire un objet (moins général) dans une représentation la plus générale qu'est l'entité. La raison de cette nécessité est la politique de la représentation des concepts de l'univers du discours que nous avons choisie.

Les transformations que nous utiliserons (voir *[Hainaut 86a]* P60-62), sont:

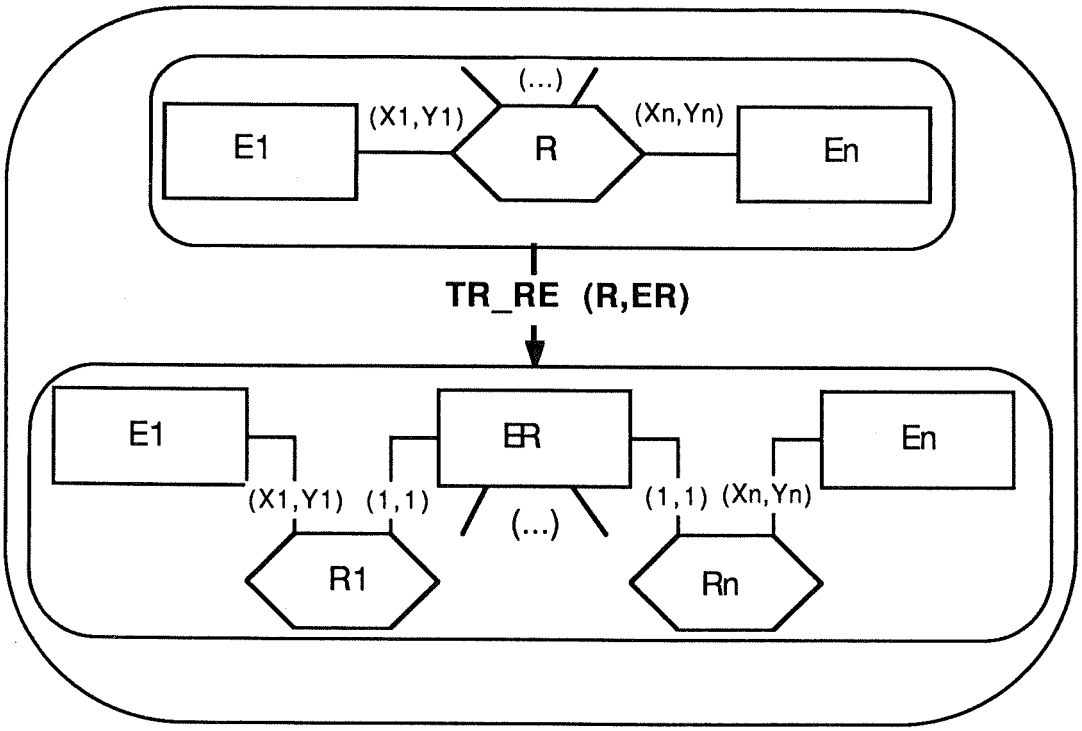
- la transformation d'une association en une entité,
- la transformation d'un attribut d'une entité en une entité, et
- la transformation d'un attribut d'une association en une entité.

Tr_RE(R,ER): Transformation d'une association

soit l'association R(E1 [X1,Y1], . . . , En [Xn,Yn]) que nous désirons transformer en une entité (voir Figure 5–4). Pour appliquer cette transformation, il suffit de réaliser les actions suivantes:

1. CREER_TE(ER) avec comme libellé celui de R, et comme attribut les attributs de R,
2. CREER_TA(R1) de libellé unique, . . . , CREER_TA(Rn) de libellé unique,
3. créer un lien-R entre: ER et R1 avec les connectivités [1,1], . . . , créer un lien-R entre: ER et Rn avec les connectivités [1,1],
4. créer un lien-R entre: E1 et R1 avec les connectivités [X1,Y1], . . . , créer un lien-R entre: En et Rn avec les connectivités [Xn,Yn],
5. si R est en correspondance avec un objet O du schéma secondaire, nous modifions cette CS dans le MS par une
   — CS: EE O*Equivalent*ER si O est un objet entité;
   — CS: AE O*Equivalent*ER si O est un objet attribut;
   — CS: RE O*Equivalent*ER si O est un objet association,

Figure 5–4: Transformation d'une association en une entité



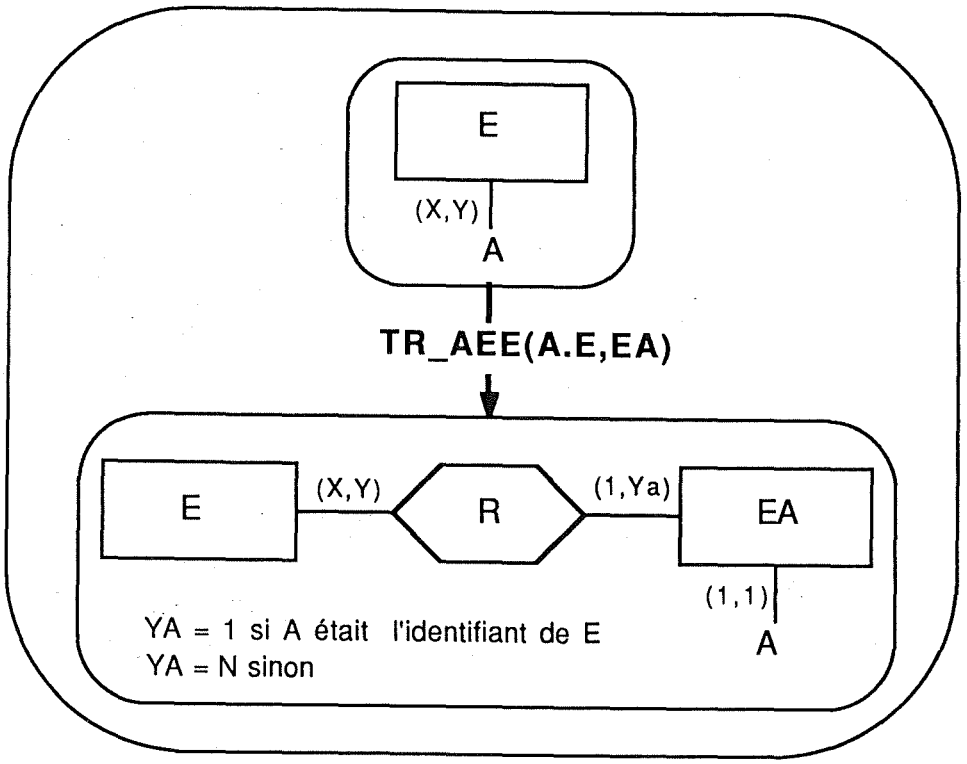
6. SUPPRIMER_TA(R),
7. la nouvelle entité ER est maintenant identifiée par les rôles que celle-ci joue dans les associations R1, . . . , Rn.

Tr_AEE(A,EA): Transformation d'un attribut d'une entité

soit l'attribut A de l'entité E (noté A.E[X,Y]) que nous désirons transformer en une entité (voir Figure 5–5). Pour appliquer cette transformation, il suffit de réaliser les actions suivantes:

1. CREER_TE(EA) avec comme libellé celui de E, et l'attribut A de connectivité [1,1],
2. CREER_TA(R) de libellé unique,
3. créer un lien-R entre: E et R avec les connectivités [X,Y],
4. créer un lien-R entre: EA et R avec les connectivités [1,Ya]: où Ya = 1 si A était l'identifiant de E, sinon Ya = N,
5. si A est en correspondance avec un attribut A1 de l'objet O du schéma secondaire, nous modifions la CS: AA A1.O*Equivalent*A.E dans le MS par la correspondance CS: AE A1.O*Equivalent*EA,
6. SUPPRIMER_ATT(A,E).

Figure 5–5: Transformation d'un attribut d'une entité en une entité



Tr_ARE(A,EA): Transformation d'un attribut d'une association

soit l'attribut A de l'association R (noté A.R[X,Y]) que l'on désire transformer en une entité. Pour appliquer cette transformation, il suffit de réaliser les actions suivantes:

1. nous réalisons la transformation Tr_RE(R,ER),
2. nous réalisons la transformation Tr_AEE(A.ER,EA) où par la première transformation A est devenu un attribut de l'entité ER.

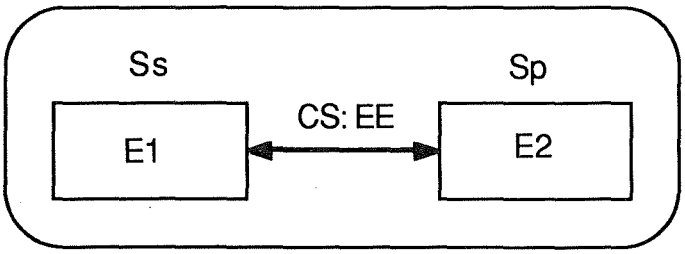
5.5.2 Intégration d'une entité

Cette section décrit les fonctions nécessaires pour intégrer une entité du schéma secondaire avec un correspondant (de n'importe quel type): nous verrons dans chaque cas quelles sont les actions à réaliser dans le schéma primaire pour intégrer la correspondance introduite.

Fonction EE: ENTITE/ENTITE

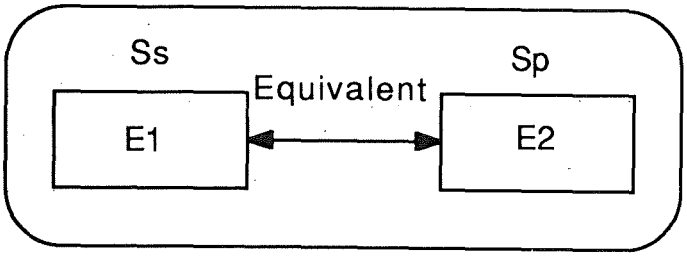
Cette fonction réalise l'intégration d'une entité de Ss avec une entité de Sp avec pour seule action l'insertion de la nouvelle correspondance dans le Mapping Structurel. Cette action n'est réalisée que si toutes les préconditions ci-dessous sont remplies.

Préconditions EE



— soit E1 une entité de Ss;
— soit E2 une entité de Sp;
— E1 n'a pas encore été mise en correspondance d'équivalence avec un objet de Sp;
— E2 n'est pas encore en correspondance d'équivalence avec un objet de Ss;
— CS: EE E1*Equivalent*E2.

Actions EE

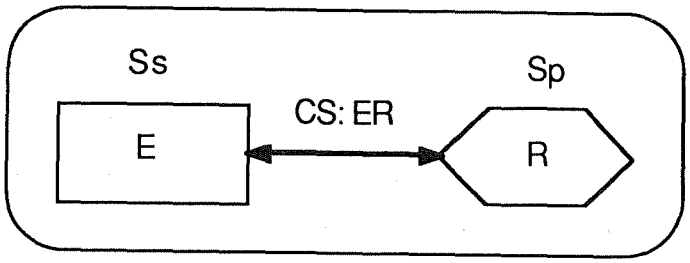


1. MS(Ss,Sp) = MS(Ss,Sp) + (CS: EE E1*Equivalent*E2),
2. l'utilisateur à la possibilité d'intervertir le libellé de E2 par le libellé de E1 et cela pour autant que le libellé de E1 soit unique parmi les libellés des entités et des associations de Sp.

Fonction ER: ENTITE/ASSOCIATION

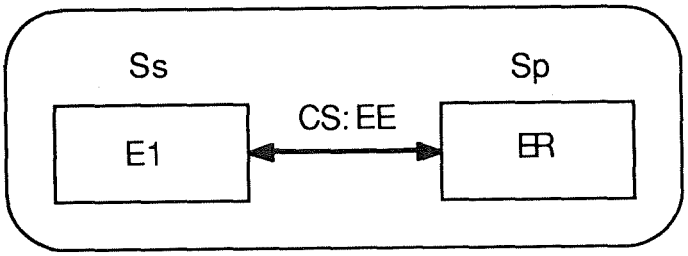
Cette fonction réalise l'intégration d'une entité de Ss avec une association de Sp avec pour actions la transformation de cette association en entité et l'insertion de la nouvelle correspondance dans le Mapping Structurel. Ces actions ne sont réalisées que si toutes les préconditions ci-dessous sont remplies.

Préconditions ER



— soit E une entité de Ss;
— soit R une association de Sp;
— E n'a pas encore été mise en correspondance d'équivalence avec un objet de Sp;
— R n'est pas encore en correspondance d'équivalence avec un objet de Ss;
— CS: ER E*Equivalent*R.

Actions ER

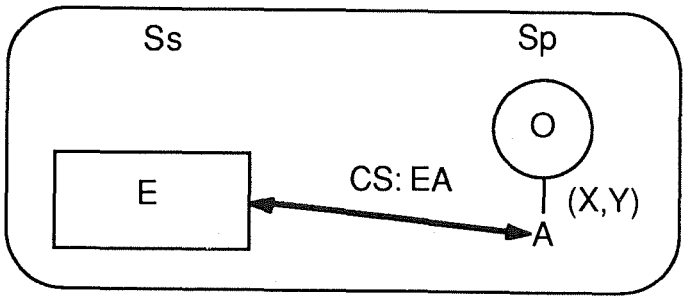


1. nous transformons l'association R en une entité ER, par l'exécution de la fonction de transformation Tr_RE(R,ER) (décrite à la section 5.5.1),
2. nous nous ramenons ainsi au cas d'une fonction EE, où nous désirons maintenant établir la CS: E*Equivalent*ER par l'application de la fonction EE.

Fonction EA: ENTITE/ATTRIBUT

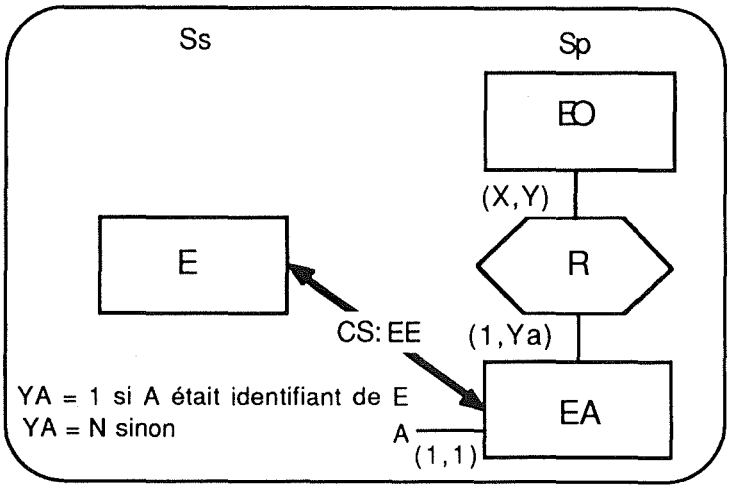
Cette fonction réalise l'intégration d'une entité de Ss avec un attribut de Sp avec pour actions la transformation de cet attribut en entité et l'insertion de la nouvelle correspondance dans le Mapping Structurel. Ces actions ne sont réalisées que si toutes les préconditions ci-dessous sont remplies.

Préconditions EA



— soit E une entité de Ss;
— soit A.O[X,Y] un attribut de l'objet O de Sp;
— E n'a pas encore été mise en correspondance d'équivalence avec un objet de Sp;
— A n'est pas encore en correspondance d'équivalence avec un objet de Ss;
— CS: EA E*Equivalent*A.O.

Actions EA



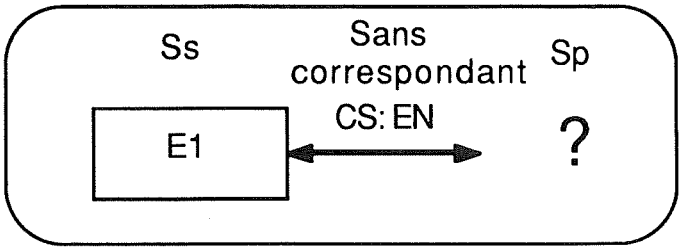
1. nous transformons l'attribut de l'objet O en une entité EA, par l'exécution de la fonction de transformation:
   1. si l'attribut n'est pas un attribut de niveau 1[9], nous appliquons d'abord la transformation Tr_AE jusqu'à ce que l'attribut A soit de niveau 1,
   2. si O est un objet de type:
      — entité, nous réalisons la transformation Tr_AE(A,EA);
      — association, nous réalisons la transformation Tr_ARE(A,EA),
2. nous nous ramenons ainsi au cas d'une fonction EE, où nous désirons maintenant établir la CS: E*Equivalent*EA par l'application de la fonction EE.

[9] Nous entendons par un attribut de niveau 1 un attribut qui a pour père soit une entité, soit une association. Par contre un attribut de niveau supérieur à 1 est un attribut dont le père est un autre attribut.

Fonction EN: ENTITE/?

Cette fonction réalise l'intégration d'une entité de Ss sans correspondant dans Sp avec pour actions la création d'une entité dans le schéma primaire et l'insertion de la nouvelle correspondance dans le Mapping Structurel. Ces actions ne sont réalisées que si toutes les préconditions ci-dessous sont remplies.

Préconditions EN



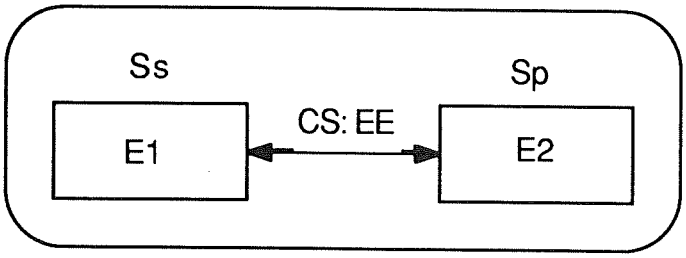
— soit E1 une entité de Ss;

— E1 n'a pas encore été mise en correspondance d'équivalence avec un objet de Sp;

— E1 n'a pas de correspondant équivalent dans Sp;

— CS: EE E1.

Actions EN



1. CREER_TE(E2), de libellé E1 si celui-ci est unique parmi les libellés des entités et des associations de Sp, sinon l'utilisateur est obligé de rentrer un libellé unique,
2. nous nous ramenons ainsi au cas d'une fonction EE, où nous désirons maintenant établir la CS: E1*Equivalent*E2 par l'application de la fonction EE.

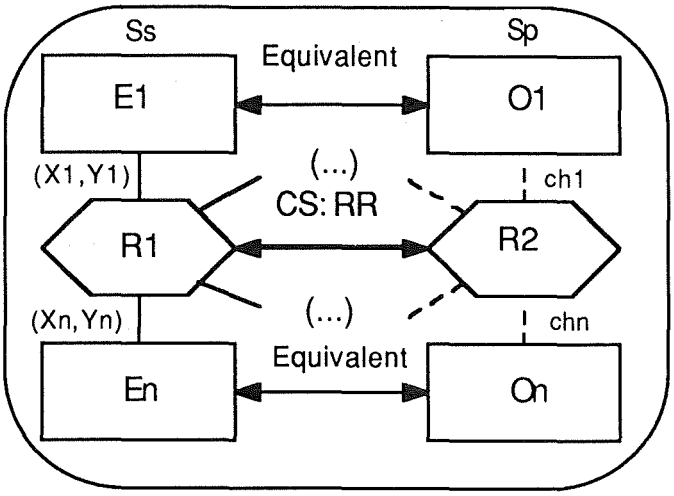
5.5.3 Intégration d'une association

Cette section décrit les fonctions nécessaires pour intégrer une association du schéma secondaire avec un correspondant (de n'importe quel type): nous verrons dans chaque cas quelles sont les actions à réaliser dans le schéma primaire pour intégrer la correspondance introduite.

Fonction RR: ASSOCIATION/ASSOCIATION

Cette fonction réalise l'intégration d'une association de Ss avec une association de Sp avec pour seule action l'insertion de la nouvelle correspondance dans le Mapping Structurel. Cette action n'est réalisée que si toutes les préconditions ci-dessous sont remplies.

Préconditions RR



— soit R1 (E1[X1,Y1], . . . , En[Xn,Yn]) une association de Ss, où E1, . . . , En sont les entités qui participent à l'association R1, avec leurs connectivités respectives;

— soit R2 une association de Sp;

— soit O1, . . . , On n objets distincts (en fait, des entités[10]) de Sp;

— R1 n'a pas encore été mise en correspondance d'équivalence avec un objet de Sp;

— R2 n'est pas encore en correspondance d'équivalence avec un objet de Ss;

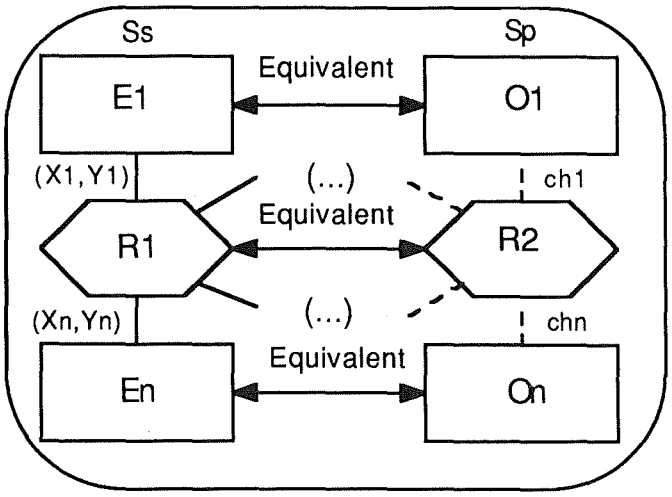
— E1 est en correspondance d'équivalence avec O1 (CS: EE E1*Equivalent*O1), . . . , En est en correspondance d'équivalence avec On (CS: EE En*Equivalent*On);

— ch1 = chemin(R2,O1) de longueur ≥ 1, . . . , chn = chemin(R2,On) de longueur ≥ 1;

— CS: RR R1*Equivalent*R2.

[10] Ces objets sont tous des entités par le choix d'une politique générale: nous gardons l'objet le plus général dans le schéma primaire en cas de conflit de structures entre les objets que nous mettons en correspondance.

Actions RR



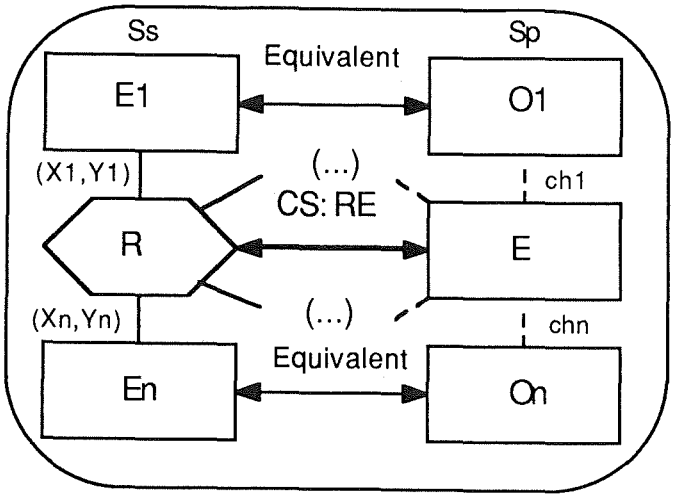
1. MS(Ss,Sp) = MS(Ss,Sp) + (CS: RR R1*Equivalent*R2),
2. pour chaque chi ($1 \leq i \leq n$) de longueur égale à 1 (c-à-d que Oi participe à l'association R2 avec [Xoi,Yoi] pour connectivités), nous modifions les connectivités[11] de la façon suivante:
   - Xoi = min(Xi,Xoi);
   - Yoi = max(Yi,Yoi),
3. l'utilisateur à la possibilité d'intervertir le libellé de R2 par le libellé de R1 et cela pour autant que le libellé de R1 soit unique parmi les libellés des entités et des associations de Sp.

Fonction RE: ASSOCIATION/ENTITE

Cette fonction réalise l'intégration d'une association de Ss avec une entité de Sp avec pour seule action l'insertion de la nouvelle correspondance dans le Mapping Structurel. Cette action n'est réalisée que si toutes les préconditions ci-dessous sont remplies.

[11] Les modifications des connectivités pour des chemins de longueur supérieure à 1 sont laissées à charge de l'utilisateur.

Préconditions RE



— soit R (E1[X1,Y1], . . . , En[Xn,Yn]) une association de Ss, où E1, . . . , En sont les entités qui participent à l'association R, avec leurs connectivités respectives;

— soit E une entité de Sp;

— soit O1, . . . , On n objets distincts (de type entité) de Sp;

— R n'a pas encore été mise en correspondance d'équivalence avec un objet de Sp;

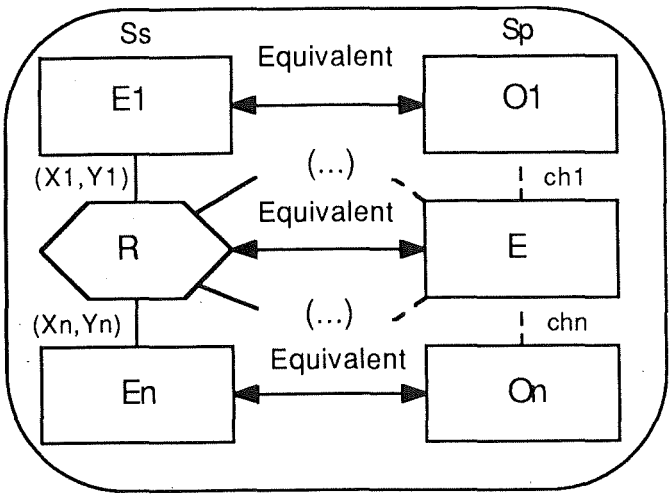
— E n'est pas encore en correspondance d'équivalence avec un objet de Ss;

— E1 est en correspondance d'équivalence avec O1 (CS: EE E1*Equivalent*O1), . . . , En est en correspondance d'équivalence avec On (CS: EE En*Equivalent*On);

— ch1 = chemin(E,O1) de longueur ≥1, . . . , chn = chemin(E,On) de longueur ≥1;

— CS: RE R*Equivalent*E.

Actions RE

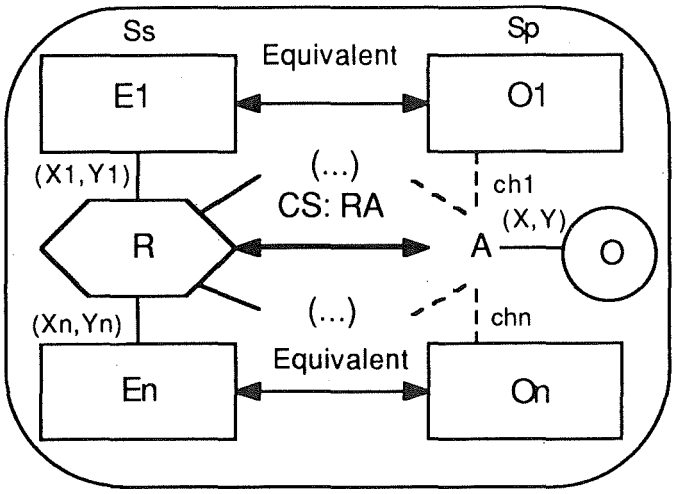


1. MS(Ss,Sp) = MS(Ss,Sp) + (CS: RE R*Equivalent*E)
2. l'utilisateur à la possibilité d'intervertir le libellé de E par le libellé de R et cela pour autant que le libellé de R soit unique parmi les libellés des entités et des associations de Sp[12].

Fonction RA: ASSOCIATION/ATTRIBUT

Cette fonction réalise l'intégration d'une association de Ss avec un attribut de Sp avec pour actions la transformation de l'attribut en entité et l'insertion de la nouvelle correspondance dans le Mapping Structurel. Ces actions ne sont réalisées que si toutes les préconditions ci-dessous sont remplies.

Préconditions RA



— soit R (E1[X1,Y1], . . . , En[Xn,Yn]) une association de Ss, où E1, . . . , En sont les entités qui participent à l'association R1, avec leurs connectivités respectives;

[12] Les modifications des connectivités des chemins ch1, . . . , chn sont laissées à charge de l'utilisateur.

— soit A.O[X,Y] un attribut de l'objet O (de type entité) de Sp;

— soit O1, . . . , On n objets distincts de Sp;

— R n'a pas encore été mise en correspondance d'équivalence avec un objet de Sp;

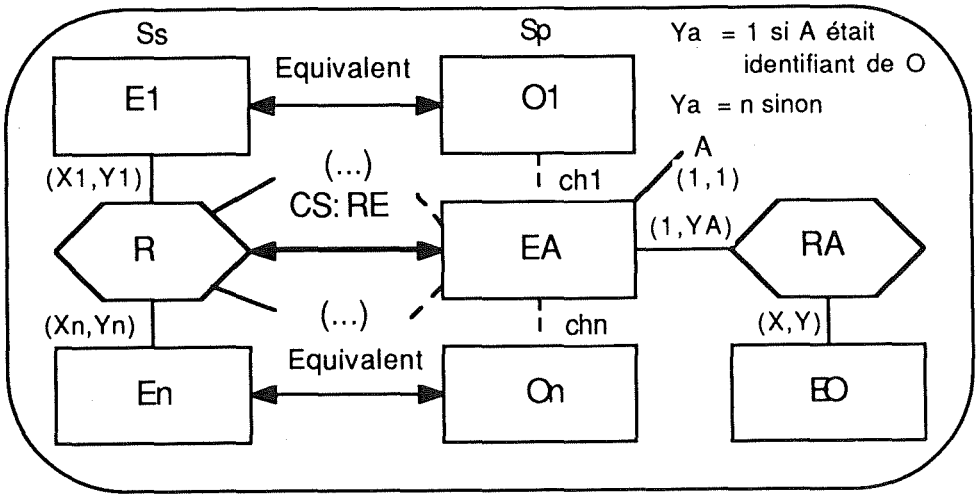
— A n'est pas encore en correspondance d'équivalence avec un objet de Ss;

— E1 est en correspondance d'équivalence avec O1 (CS: EE E1*Equivalent*O1), . . . , En est en correspondance d'équivalence avec On (CS: EE En*Equivalent*On);

— Il existe au moins un chemin: ch1 = chemin(A,O1), . . . , chn = chemin(A,On) de longueur ≥1;

— CS: RA R*Equivalent*A.O.

Actions RA

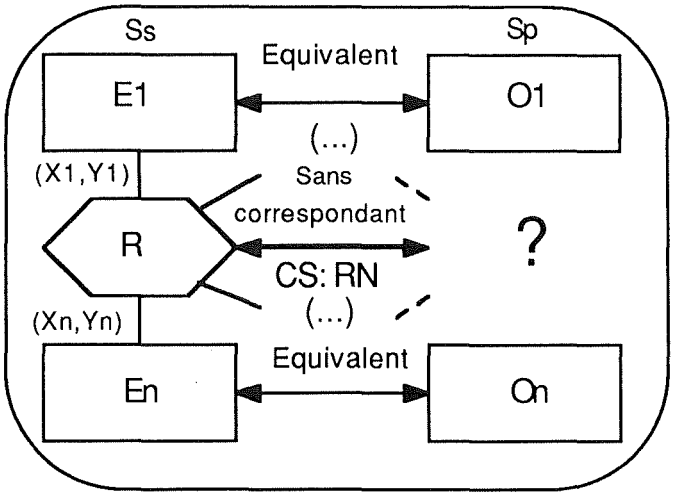


1. nous transformons l'attribut de l'objet O en une entité EA, par l'exécution de la fonction de transformation:
   1. si l'attribut n'est pas un attribut de niveau 1[9], nous appliquons d'abord la transformation Tr_AE jusqu'à ce que l'attribut A soit de niveau 1,
   2. si O est un objet de type:
      — entité, nous réalisons la transformation Tr_AE(A,EA);
      — association, nous réalisons la transformation Tr_ARE(A,EA),
2. nous nous ramenons ainsi au cas d'une fonction RE, où nous désirons maintenant établir la CS: R*Equivalent*EA par l'application de la fonction RE.

Fonction RN: ASSOCIATION/?

Cette fonction réalise l'intégration d'une association de Ss sans correspondant dans Sp avec pour actions la création d'une association dans Sp et l'insertion de la nouvelle correspondance dans le Mapping Structurel. Ces actions ne sont réalisées que si toutes les préconditions ci-dessous sont remplies.

Préconditions RN



— soit R1 (E1[X1,Y1], . . . , En[Xn,Yn]) une association de Ss, où E1, . . . , En sont les entités qui participent à l'association R1, avec leurs connectivités respectives;

— soit O1, . . . , On n objets distincts de Sp;

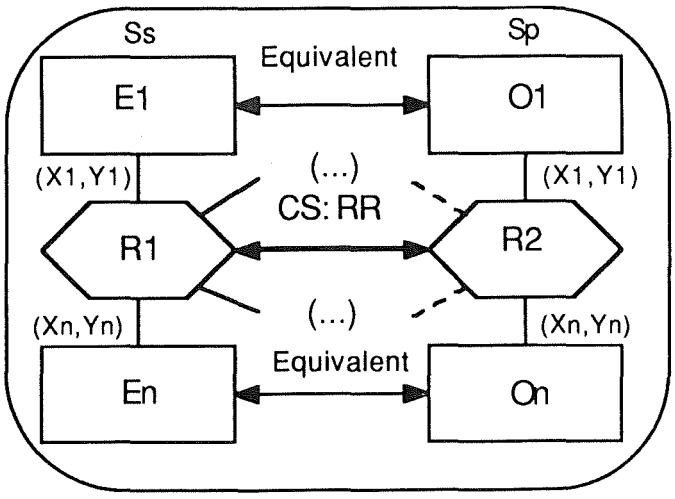
— R1 n'a pas encore été mise en correspondance d'équivalence avec un objet de Sp;

— R1 n'a pas de correspondant équivalent dans Sp,

— E1 est en correspondance d'équivalence avec O1 (CS: EE E1*Equivalent*O1), . . . , En est en correspondance d'équivalence avec On (CS: EE En*Equivalent*On);

— CS: RN R1.

Actions RN



1. CREER_TA(R2), de libellé R1 si celui-ci est unique parmi les libellés des entités et des associations de Sp, sinon l'utilisateur est obligé de rentrer un libellé unique

2. créer un lien-R entre: O1 et R2 avec les connectivités [X1,Y1], . . . , créer un lien-R entre: On et Rn avec les connectivités [Xn,Yn],
3. nous nous ramenons ainsi au cas d'une fonction RR, où nous voulons maintenant établir la CS: R1*Equivalent*R2 par l'application de la fonction RR.

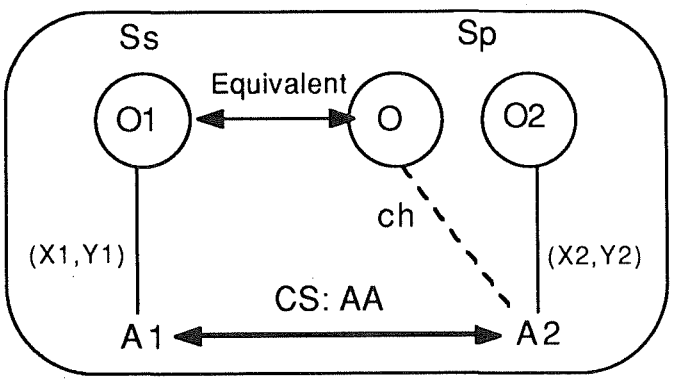
5.5.4 Intégration d'un attribut

Cette section décrit les fonctions nécessaires pour intégrer un attribut du schéma secondaire avec un correspondant (de n'importe quel type): nous verrons dans chaque cas quelles sont les actions à réaliser dans le schéma primaire pour intégrer la correspondance introduite.

Fonction AA: ATTRIBUT/ATTRIBUT

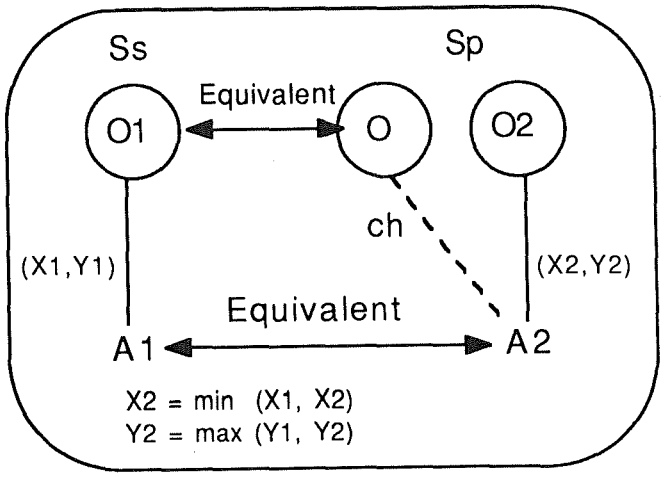
Cette fonction réalise l'intégration d'un attribut de Ss avec un attribut de Sp avec pour seule action l'insertion de la nouvelle correspondance dans le Mapping Structurel. Cette action n'est réalisée que si toutes les préconditions ci-dessous sont remplies.

Préconditions AA



— soit A1.O1[X1,Y1] un attribut de l'objet O1 de Ss;
— soit A2.O2[X2,Y2] un attribut de l'objet O2 de Sp;
— soit O un objet de Sp;
— A1 n'a pas encore été mis en correspondance d'équivalence avec un objet de Sp;
— A2 n'est pas encore en correspondance d'équivalence avec un objet de Ss;
— O1 est en correspondance d'équivalence avec O;
— ch = chemin(A2,O) de longueur ≥ 1;
— CS: AA A1.O1*Equivalent*A2.O2.

Actions AA

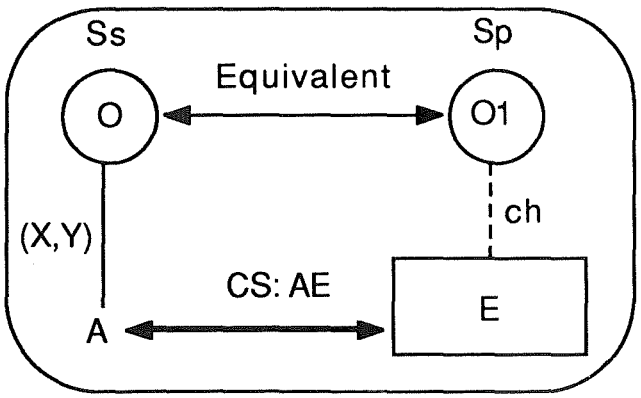


1. MS(Ss,Sp) = MS(Ss,Sp) + (CS: AA A1.O1*Equivalent*A2.O2),
2. les connectivités de A2 dans O2 sont maintenant:
   — X2 = min(X1,X2);
   — Y2 = max(Y1,Y2),
3. l'utilisateur à la possibilité d'intervertir le libellé de A2 par le libellé de A1 et cela pour autant que le libellé de A1 soit unique parmi les libellés des attributs de même niveau de l'objet O2 de Sp.

Fonction AE: ATTRIBUT/ENTITE

Cette fonction réalise l'intégration d'un attribut de Ss avec une entité de Sp avec pour seule action l'insertion de la nouvelle correspondance dans le Mapping Structurel. Cette action n'est réalisée que si toutes les préconditions ci-dessous sont remplies.

Préconditions AE



— soit A.O[X,Y] un attribut de l'objet O de Ss;
— soit E une entité de Sp;
— soit O1 un objet de Sp;

— A n'a pas encore été mis en correspondance d'équivalence avec un objet de Sp;

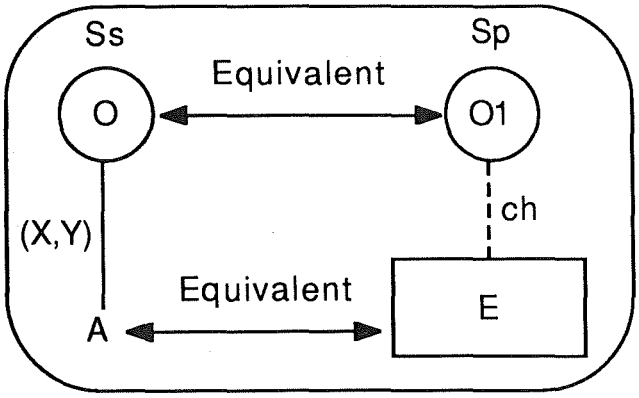
— E n'est pas encore en correspondance d'équivalence avec un objet de Ss;

— O est en correspondance d'équivalence avec O1;

— ch = chemin(E,O1) de longueur ≥1,

— CS: AE A.O*Equivalent*E.

Actions AE

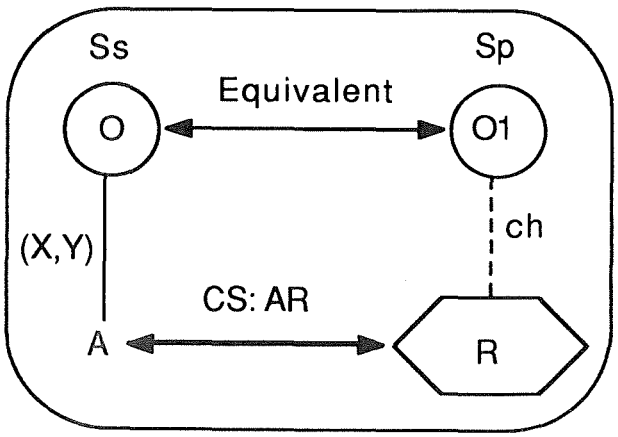


1. MS(Ss,Sp) = MS(Ss,Sp) + (CS: AE A.O*Equivalent*E),
2. l'utilisateur à la possibilité d'intervertir le libellé de E par le libellé de A et cela pour autant que le libellé de A soit unique parmi les libellés des entités et des associations de Sp.

Fonction AR: ATTRIBUT/ASSOCIATION

Cette fonction réalise l'intégration d'un attribut de Ss avec une association de Sp avec pour seule action l'insertion de la nouvelle correspondance dans le Mapping Structurel. Cette action n'est réalisée que si toutes les préconditions ci-dessous sont remplies.

Préconditions AR



— soit A.O[X,Y] un attribut de l'objet O de Ss;

— soit R une association de Sp;

— soit O1 un objet de Sp;

— A n'a pas encore été mis en correspondance d'équivalence avec un objet de Sp;

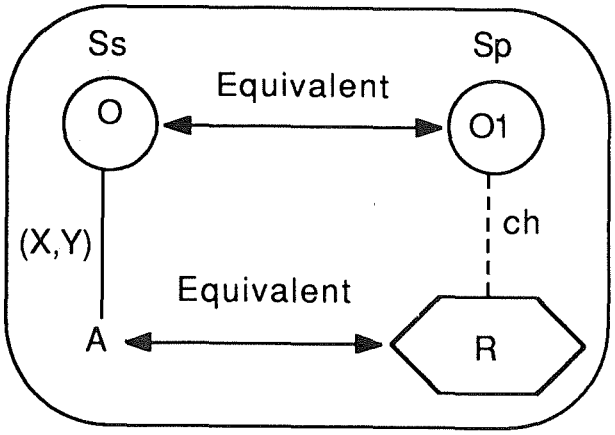
— R n'est pas encore en correspondance d'équivalence avec un objet de Ss;

— O est en correspondance d'équivalence avec O1;

— ch = chemin(R,O1) de longueur ≥ 1;

— CS: AR A.O*Equivalent*R.

Actions AR

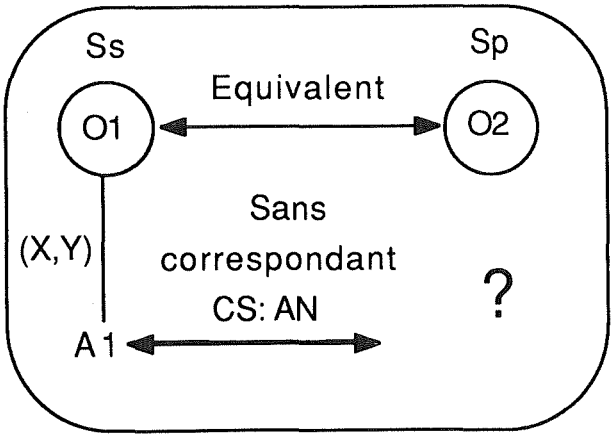


1. MS(Ss,Sp) = MS(Ss,Sp) + (CS: AR A.O*Equivalent*R),
2. si le chemin ch a une longueur égale à 1 avec pour connectivités [Xo1,Yo1], nous modifions les connectivités[11] de la façon suivante:
   — Xo1 = min(X,Xo1);
   — Yo1 = max(Y,Yo1),
3. l'utilisateur à la possibilité d'intervertir le libellé de R par le libellé de A et cela pour autant que le libellé de A soit unique parmi les libellés des entités et des associations de Sp.

Fonction AN: ATTRIBUT/?

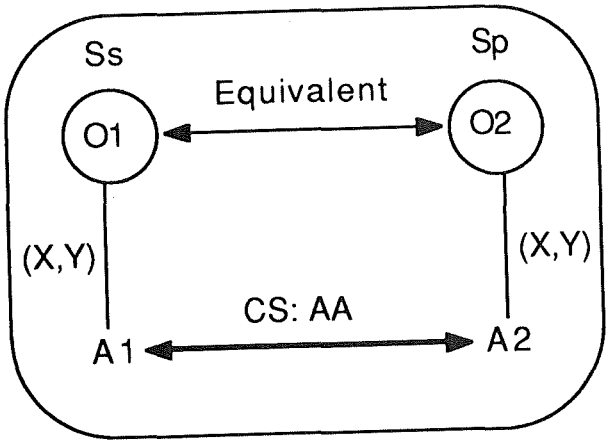
Cette fonction réalise l'intégration d'un attribut de Ss sans correspondant dans Sp avec pour actions la creéation de l'attribut dans le schéma primaire et l'insertion de la nouvelle correspondance dans le Mapping Structurel. Ces actions ne sont réalisées que si toutes les préconditions ci-dessous sont remplies.

Préconditions AN



— soit A1.O1[X,Y] un attribut de l'objet O1 de Ss;
— soit O2 un objet de Sp;
— A1 n'a pas encore été mis en correspondance d'équivalence avec un objet de Sp;
— A1 n'a pas de correspondant équivalent dans Sp;
— O1 est en correspondance d'équivalence avec O2;
— CS: AN A1.O1.

Actions AN



1. CREER_ATT(A2,O2),
2. créer un lien-A entre: O2 et A2 avec les connectivités [X,Y],
3. nous nous ramenons ainsi au cas d'une fonction AA, où nous voulons maintenant établir la CS: A1.O1*Equivalent*A2.O2 par l'application de la fonction AA.

5.6 Algorithme général d'intégration

Cette section présente un algorithme d'intégration avec les différentes fonctions détaillées à la section 5.5 qui sont autant d'algorithmes destinés à intégrer des vues de bases de données non encore existantes. Afin de respecter les choix concernant notre méthodologie, cet algorithme sera interactif, il supposera que les schémas à intégrer sont corrects et se basera sur les correspondances introduites au fur et à mesure par l'utilisateur.

5.6.1 L'algorithme

Une fois que l'utilisateur a demandé de lancer le processus d'intégration, l'algorithme suivant s'exécute:

1. Introduction du schéma à intégrer ou abandon,
2. Introduction du schéma à enrichir ou abandon,
3. Initialisation de la liste des correspondances,
4. Tant Que l'utilisateur ne demande pas l'arrêt du processus d'intégration
   Faire
   5. Introduire l'élément du schéma secondaire à intégrer,
   6. Dire s'il a un correspondant ou annuler la commande,
   7. S'il a un correspondant, le choisir dans le schéma primaire,
   8. Intégrer,
9. Demander la sauvegarde du mapping structurel.

La fonction "INTEGRER" consiste à détecter quels objets ont été sélectionnés, s'ils vérifient les préconditions et applique la fonction d'intégration correspondante.

5.6.2 La fonction "INTEGRER"

Nous utilisons la même notation que pour les correspondances: par exemple "EE" signifiera que l'utilisateur a sélectionné dans le schéma secondaire (Ss) un objet qui est une entité et dans le schéma primaire (Sp) un objet qui est aussi une entité.

Cette fonction consiste en l'algorithme suivant:

1. Si l'utilisateur a choisi EE,
   Alors EE_intégration;
2. Si l'utilisateur a choisi ER,
   Alors ER_intégration;
3. Si l'utilisateur a choisi EA,
   Alors EA_intégration;
4. Si l'utilisateur a choisi EN,
   Alors EN_intégration;
5. Si l'utilisateur a choisi RR,
   Alors RR_intégration;
6. Si l'utilisateur a choisi RE,
   Alors RE_intégration;

7. Si l'utilisateur a choisi RA,
   Alors RA_intégration;
8. Si l'utilisateur a choisi RN,
   Alors RN_intégration;
9. Si l'utilisateur a choisi AA,
   Alors AA_intégration;
10. Si l'utilisateur a choisi AE,
    Alors AE_intégration;
11. Si l'utilisateur a choisi AR,
    Alors AR_intégration;
12. Si l'utilisateur a choisi AN,
    Alors AN_intégration.

La fonction EE_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction EE dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction EE dans la section 5.5,
2. Pour chaque attribut de l'entité à intégrer,
   Faire
   3. Dire s'il a un correspondant ou annuler la commande,
   4. S'il a un correspondant, le choisir,
   5. Intégrer.

Si l'utilisateur décide d'annuler, il pourra revenir par après à l'intégration des attributs en sélectionnant les attributs de l'entité qu'il a décidé de ne pas intégrer dans un premier temps.

La fonction ER_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction ER dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction ER dans la section 5.5,
2. Pour chaque attribut de l'entité à intégrer,
   Faire
   3. Dire s'il a un correspondant ou annuler la commande,
   4. S'il a un correspondant, le choisir,
   5. Intégrer.

Si l'utilisateur décide d'annuler, il pourra revenir par après à l'intégration des attributs en sélectionnant les attributs de l'entité qu'il a décidé de ne pas intégrer dans un premier temps.

La fonction EA_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction EA dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction EA dans la section 5.5,
2. Pour chaque attribut de l'entité à intégrer,
   Faire
   3. Dire s'il a un correspondant ou annuler la commande,
   4. S'il a un correspondant, le choisir,
   5. Intégrer.

Si l'utilisateur décide d'annuler, il pourra revenir par après à l'intégration des attributs en sélectionnant les attributs de l'entité qu'il a décidé de ne pas intégrer dans un premier temps.

La fonction EN_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction EN dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction EN dans la section 5.5,
2. Pour chaque attribut de l'entité à intégrer,
   Faire
   3. Dire s'il a un correspondant ou annuler la commande,
   4. S'il a un correspondant, le choisir,
   5. Intégrer.

Si l'utilisateur décide d'annuler, il pourra revenir par après à l'intégration des attributs en sélectionnant les attributs de l'entité qu'il a décidé de ne pas intégrer dans un premier temps.

La fonction RR_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction RR dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie, pour la fonction RR dans la section 5.5,
2. Pour chaque attribut de l'association à intégrer,
   Faire
   3. Dire s'il a un correspondant ou annuler la commande,
   4. S'il a un correspondant, le choisir,
   5. Intégrer.

Si l'utilisateur décide d'annuler, il pourra revenir par après à l'intégration des attributs en sélectionnant les attributs de l'association qu'il a décidé de ne pas intégrer dans un premier temps.

La fonction RE_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction RE dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction RE dans la section 5.5,
2. Pour chaque attribut de l'association à intégrer,
   Faire
   3. Dire s'il a un correspondant ou annuler la commande,
   4. S'il a un correspondant, le choisir,
   5. Intégrer.

Si l'utilisateur décide d'annuler, il pourra revenir par après à l'intégration des attributs en sélectionnant les attributs de l'association qu'il a décidé de ne pas intégrer dans un premier temps.

La fonction RA_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction RA dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction RA dans la section 5.5,
2. Pour chaque attribut de l'association à intégrer,
   Faire
   3. Dire s'il a un correspondant ou annuler la commande,
   4. S'il a un correspondant, le choisir,
   5. Intégrer.

Si l'utilisateur décide d'annuler, il pourra revenir par après à l'intégration des attributs en sélectionnant les attributs de l'association qu'il a décidé de ne pas intégrer dans un premier temps.

La fonction RN_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction RN dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction RN dans la section 5.5,
2. Pour chaque attribut de l'association à intégrer,
   Faire
   3. Dire s'il a un correspondant ou annuler la commande,

4. S'il a un correspondant, le choisir,
5. Intégrer.

Si l'utilisateur décide d'annuler, il pourra revenir par après à l'intégration des attributs en sélectionnant les attributs de l'association qu'il a décidé de ne pas intégrer dans un premier temps.

La fonction AA_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction AA dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction AA dans la section 5.5,

La fonction AE_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction AE dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction AE dans la section 5.5.

La fonction AR_INTEGRATION

Cela correspond à l'algorithme:

1. Si au moins une des préconditions définies pour la fonction AR dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction AR dans la section 5.5.

La fonction AN_INTEGRATION

Cela correspond à l'algorithme:

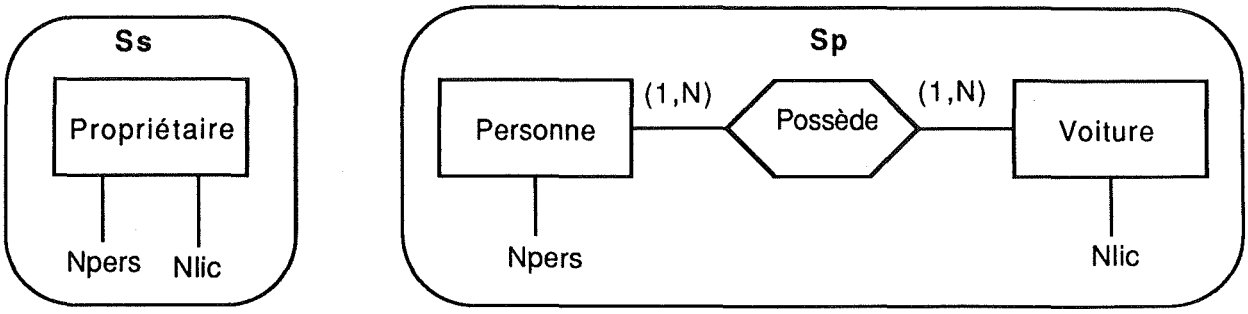
1. Si au moins une des préconditions définies pour la fonction AN dans la section 5.5 n'est pas vérifiée,
   Alors afficher message d'erreur et abandon de la commande;
   Sinon exécuter la partie action définie pour la fonction AN dans la section 5.5.

5.7 Quelques exemples

Cette section a pour but de montrer le fonctionnement de l'algorithme d'intégration énoncé à la section 5.6 sur base de quelques cas démonstratifs. Le premier exemple (tiré de *[Larson & Al 89]*) montre comment nous réalisons l'intégration d'une entité et d'une association, ainsi que les modifications que l'utilisateur peut réaliser sur le schéma intégré. Le deuxième exemple (tiré de *[Spaccapietra & Parent 90]*) montre comment nous élaborons l'intégration d'une entité et d'un attribut. Le dernier exemple montre la manière d'intégrer trois schémas un peu plus "complexes".

Propriétaire de voiture

Etape 0 Soit les schémas (Ss et Sp) suivants:



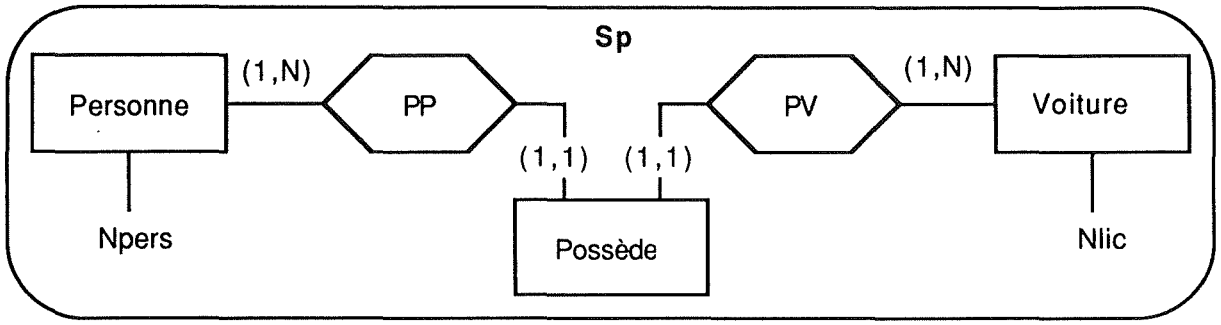
Nous désirons intégrer Ss dans Sp (du schéma le plus compact vers le schéma le plus éclaté). Au début de l'intégration de Ss dans Sp, l'ensemble MS(Ss,Sp) est vide.

Etape 1 La seule correspondance que nous pouvons établir pour l'instant est une correspondance:

```
CS : ER Propriétaire*Equivalent*Possède
```

(car nous ne pouvons commencer l'intégration par les attributs de l'entité Propriétaire, puisque cette dernière n'a pas encore été mise en correspondance avec un objet de Sp).

Les préconditions de la fonction ER d'intégration sont vérifiées et l'application des actions de la fonction ER sur le schéma Sp donne le résultat suivant:



A la fin de cette première étape, nous avons dans MS(Ss,Sp) une correspondance:

```
MS(Ss,Sp) =
            EE Propriétaire*Equivalent*Possède
```

Etape 2 Cette étape est le déclenchement d'un sous-processus d'intégration d'une entité (à cause de la politique choisie dans notre algorithme). Nous réalisons la mise en correspondance des attributs (Npers et Nlic) de l'entité Propriétaire dans Ss.

Nous établissons deux correspondances:

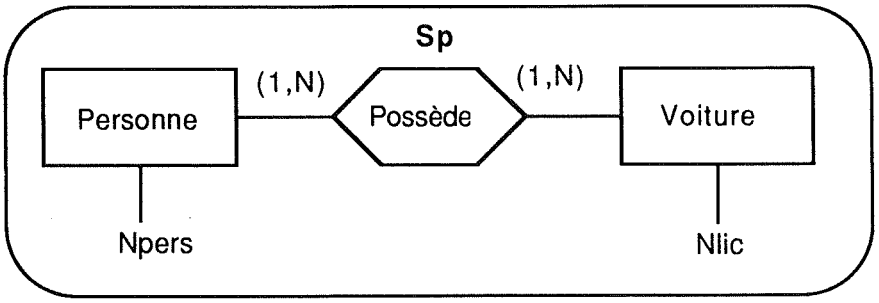
```
CS: AA Npers.Propriétaire*Equivalent*Npers.Personne
CS: AA Nlic.Propriétaire*Equivalent*Nlic.Voiture
```

Les préconditions de la fonction AA d'intégration sont vérifiées pour l'application de ces deux correspondances. Le résultat final du processus d'intégration du schéma Ss dans le schéma Sp est identique à celui obtenu à la fin de l'étape 1 (c'est-à-dire qu'aucune transformation n'a

été opérée sur le schéma primaire). La seule différence est l'ajout des deux correspondances dans MS(Ss,Sp).

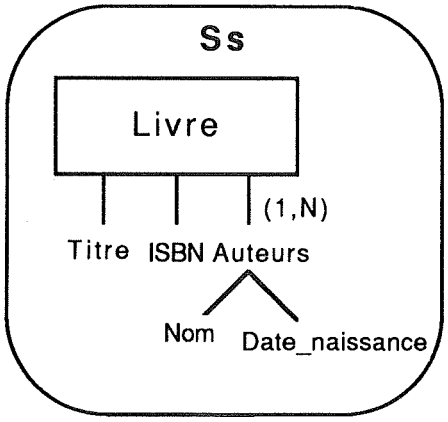
```
MS(Ss,Sp) =
            EE Propriétaire*Equivalent*Possède
            AA Npers.Propriétaire*Equivalent*Npers.Personne
            AA Nlic.Propriétaire*Equivalent*Nlic.Voiture
```

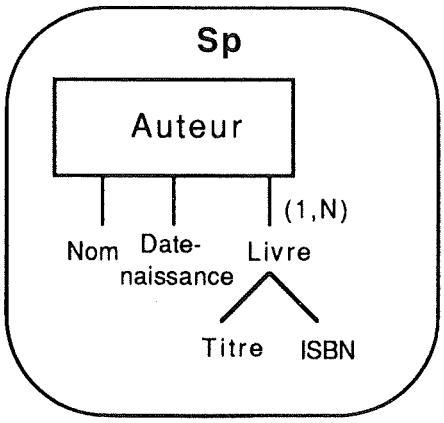
Etape complémentaire Nous constatons que la structure PP-possède-PV peut se ramener à une structure unique qu'est l'association. L'utilisateur peut transformer le schéma intégré de la façon suivante:



Système d'Information d'une bibliothèque

Etape 0 Soit les schémas (Ss et Sp) suivants:



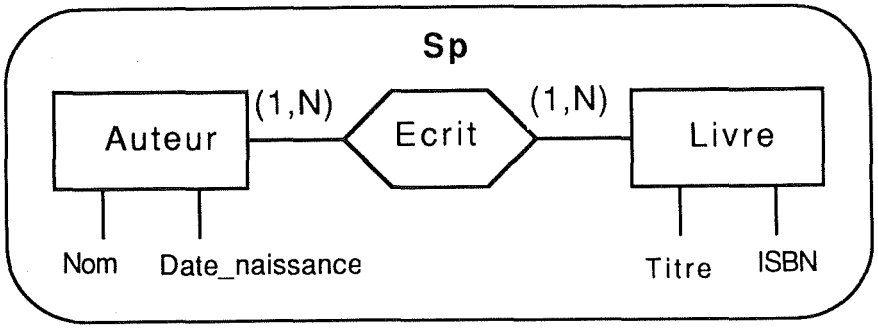


Ss représente l'univers du discours des livres, où chaque livre est décrit par son titre, son numéro ISBN, ainsi que le nom et la date de naissance des auteurs (un ou plusieurs auteurs) qui ont écrit le livre.
Sp représente l'univers du discours des auteurs, où chaque auteur est décrit par son nom, sa date de naissance ainsi que les titres et les numéros ISBN des livres qu'il a écrit.

Nous désirons intégrer Ss dans Sp. Au début de l'intégration de Ss dans Sp, l'ensemble MS(Ss,Sp) est vide.

Etape 1 La seule possibilité de mettre en correspondance un objet de Ss avec un objet de Sp, est la mise en correspondance de l'entité livre de Ss: CS: EA Livre*Equivalent*Livre.Auteur.

Les préconditions de la fonction EA d'intégration sont vérifiées et l'exécution des actions de cette fonction EA, donne le résultat suivant:



Nous avons dans MS(Ss,Sp), à la fin de l'exécution de la fonction

```
MS(Ss,Sp) =
            EE Livre*Equivalent*Livre
```

Etape 2 Suite à l'algorithme que nous nous sommes défini à la section 5.6, l'intégration d'une entité, entraîne le déclenchement d'un sous-processus d'intégration des attributs de l'entité Livre que nous venons d'intégrer dans le schéma primaire. Il faut établir une correspondance pour chaque attribut de l'entité Livre avec un objet de Sp. Nous avons donc comme correspondances pour les attributs de l'entité Livre:

```
CS: AA Titre.Livre*Equivalent*Titre.Livre
CS: AA ISBN.Livre*Equivalent*ISBN.Livre
CS: AE Auteur.Livre*Equivalent*Auteur
CS: AA Nom.Auteur*Equivalent*Nom.Auteur
CS: AA Date_naissance.Auteur*Equivalent*Date_naissance.Auteur
```

L'intégration de ces correspondances ne transforme pas le schéma primaire obtenu à la fin de la première étape. Seul l'ensemble MS(Ss,Sp) a été affecté par l'établissement de ces correspondances:

```
MS(Ss,Sp) =
            EE Livre*Equivalent*Livre
            AA Titre.Livre*Equivalent*Titre.Livre
            AA ISBN.Livre*Equivalent*ISBN.Livre
            AE Auteur.Livre*Equivalent*Auteur
            AA Nom.Auteur*Equivalent*Nom.Auteur
            AA Date_naissance.Auteur*Equivalent*Date_naissance.Auteur
```

Le lecteur pourra réaliser l'exercice inverse en intégrant le schéma Sp dans le schéma Ss (MS(Sp,Ss)). Le résultat auquel il faut arriver est identique à celui obtenu dans l'intégration de Ss dans Sp (MS(Ss,Sp)).

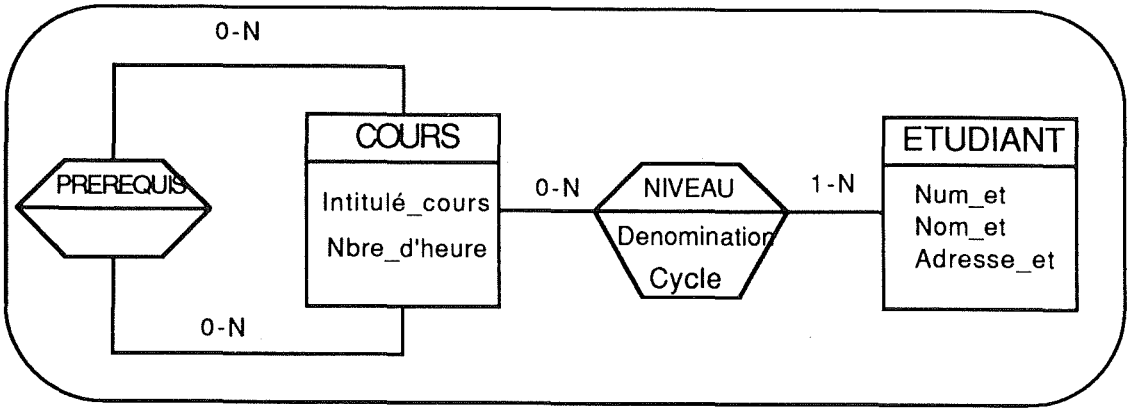
Maintenant que nous avons présenté la démarche pour des cas simples, nous abordons un exemple plus complexe qui montre comment procéder lorsqu'il y a plusieurs schémas à intégrer.

Gestion des cours à option

Pour la gestion des cours à option de l'institut d'informatique, il existe trois sous-schémas de structures de données.

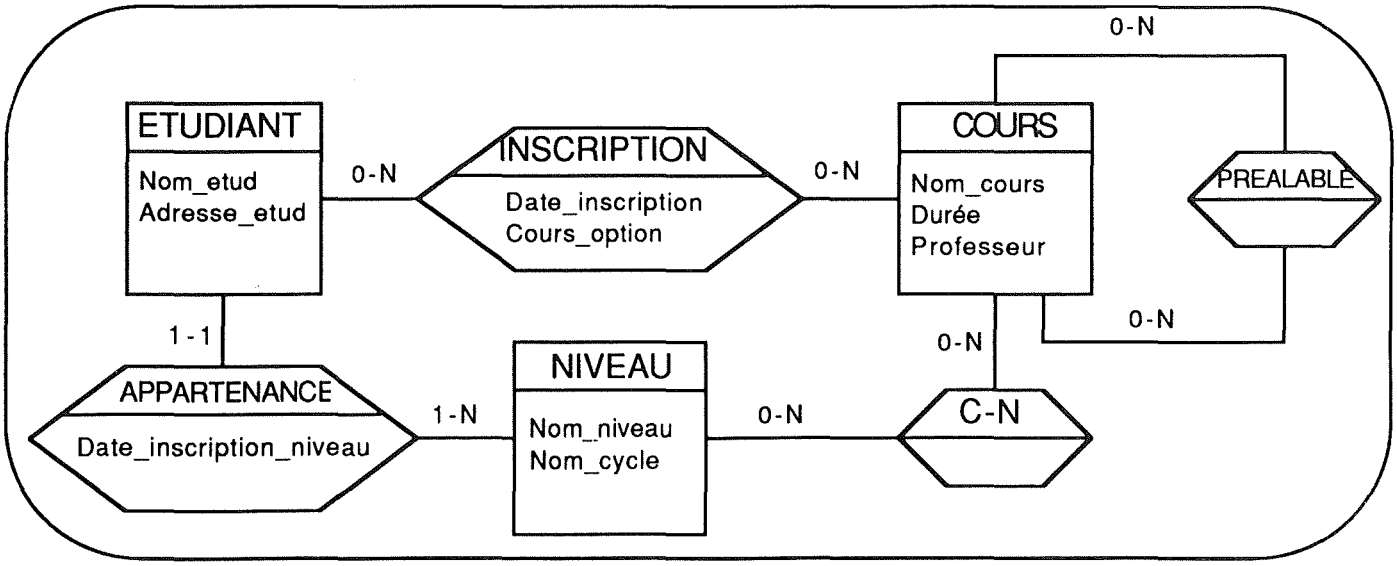
Le premier (voir Figure 5–6) indique aux étudiants les cours à option auxquels ils peuvent s'inscrire en fonction de leur niveau (exemple: 1ère licence, 2ème licence, 3ème licence, 1ère maîtrise, 2ème maîtrise) dans un cycle d'études (licence et maîtrise, maîtrise) et des cours prérequis.

Figure 5–6: Sous-schéma 1



Le second sous-schéma (voir Figure 5–7) décrit la vue du secrétariat administratif de l'institut concernant les inscriptions des étudiants aux cours à option.

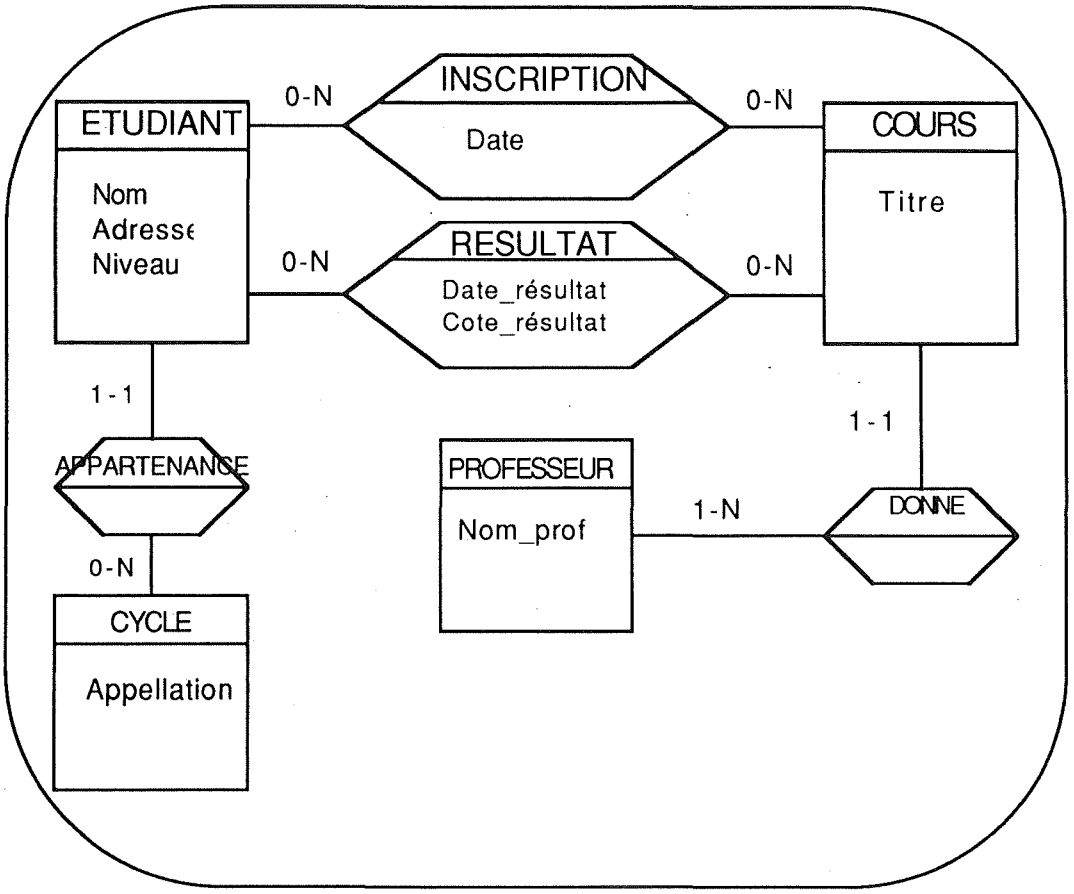
Figure 5–7: Sous-schéma 2



Ce sous-schéma doit permettre de vérifier que l'étudiant s'inscrit dans un cours à option de niveau autorisé et qu'il a suivi les cours préalables (prérequis) éventuels.

Le dernier sous-schéma (voir Figure 5–8) spécifie les informations requises par le secrétariat académique pour l'enregistrement de la cote d'un étudiant relative à un cours à option.

Figure 5–8: Sous-schéma 3



Etape 0 Nous désirons intégrer ces trois sous-schémas en un seul schéma global.

Par le choix de notre politique, nous ne pouvons intégrer que deux schémas à la fois. Nous choisissons le premier schéma comme schéma primaire pour illustrer un maximum de modifications.

Nous procéderons en deux phases:

1. Intégration du sous-schéma 2 dans le sous-schéma 1
   (MS(sous-schéma 2, sous-schéma 1))
2. Intégration du sous-schéma 3 dans le sous-schéma 2[13]
   (MS(sous-schéma 3, sous-schéma 2))

[13] A ce moment le sous-schéma 2 est la fusion des deux schémas de la Figure 5–6 et la Figure 5–7.

Etape 1 Nous appellerons le sous-schéma 1 "S1" et le sous-schéma 2 "S2". Les correspondances que nous introduisons dans une première phase sont:

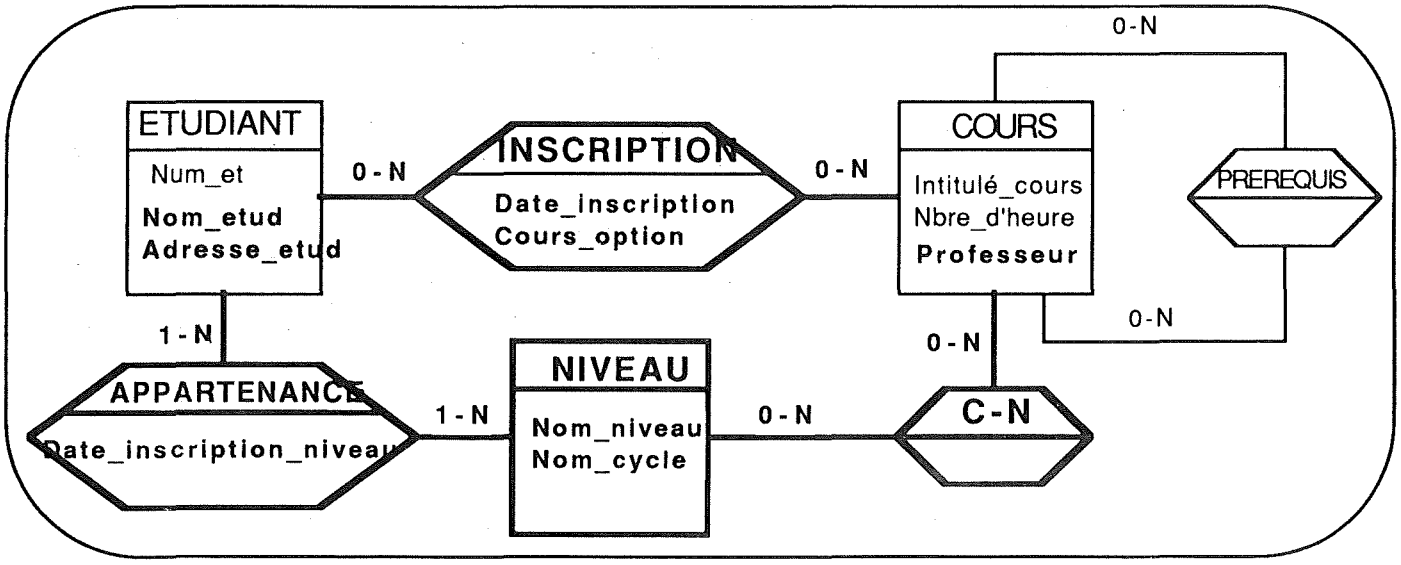
```
CS : EE COURS*Equivalent*COURS
CS : AA Nom_cours.COURS*Equivalent*Intitulé_cours.COURS
CS : AA Durée.COURS*Equivalent*Nbre_d'heure.COURS
CS : AN Professeur.COURS

CS : RR PREALABLE*Equivalent*PREREQUIS

CS : EE ETUDIANT*Equivalent*ETUDIANT
CS : AA Nom_étud.ETUDIANT*Equivalent*Nom_et.ETUDIANT
CS : AA Adresse_étud.ETUDIANT*Equivalent*Adresse_et.ETUDIANT

CS : ER NIVEAU*Equivalent*NIVEAU
CS : AA Nom_niveau.NIVEAU*Equivalent*Dénomination.NIVEAU
CS : AA Nom_cycle.NIVEAU*Equivalent*Cycle.NIVEAU

CS : RN INSCRIPTION
CS : AN Date_inscription.INSCRIPTION
CS : AN Cours_option.INSCRIPTION

CS : RR APPARTENANCE*Equivalent*R1 ❶
CS : AN Date_inscription_niveau.APPARTENANCE
CS : RR C-N*Equivalent*R2 ❷
```

❶ L'association R1 est une association créée par la transformation TR_RE (voir section 5.5.1) de NIVEAU en entité dans le schéma primaire.

❷ L'association R2 est la seconde association créée par la même transformation.

Nous pouvons voir le résultat après cette première phase à la Figure 5–9: les différences par rapport au schéma primaire (voir Figure 5–6) sont mises en gras. Les correspondances introduites dans MS(S1,S2) sont identiques à celles que nous venons d'introduire.

Figure 5–9: Le schéma intermédiaire



Etape 2 Nous intégrons maintenant le sous-schéma 3 (noté S3) dans le sous-schéma 2 modifié (toujours noté S2). Puisque nous intégrons deux "nouveaux" schémas, le mapping structurel MS(S3,S2) est réinitialisé. Les correspondances à introduire sont:

```
CS : EE COURS*Equivalent*COURS
CS : AA Titre.COURS*Equivalent*Intitulé_cours.COURS

CS : EE ETUDIANT*Equivalent*ETUDIANT
CS : AA Nom.ETUDIANT*Equivalent*Nom_etud.ETUDIANT
CS : AA Adresse.ETUDIANT*Equivalent*Adresse_etud.ETUDIANT
CS : AE Niveau.ETUDIANT*Equivalent*NIVEAU

CS : EA PROFESSEUR*Equivalent*Professeur.COURS ❸
CS : AA Nom_prof.PROFESSEUR*Equivalent*Professeur.PROFESSEUR ❹
CS : RR DONNE*Equivalent*R ❺

CS : EA CYCLE*Equivalent*Nom_cycle.NIVEAU ❻
CS : AA Appellation.CYCLE*Equivalent*Nom_cycle.CYCLE
CS : RR APPARTENANCE*Equivalent*R

CS : RR INSCRIPTION*Equivalent*INSCRIPTION
CS : AA Date.INSCRIPTION*Equivalent*Date_inscription.INSCRIPTION

CS : RN RESULTAT
CS : AN Date_résultat.RESULTAT
CS : AN Cote_résultat.RESULTAT
```

❸ Par la transformation TR_AEE (voir section 5.5.1, l'attribut Professeur.COURS est remplacé dans le schéma primaire par l'entité PROFESSEUR. Elle possède un attribut et est reliée à COURS.

❹ C'est l'attribut de PROFESSEUR créé par la même transformation.

❺ C'est l'association qui relie PROFESSEUR à COURS.

❻ Nous appliquons ici aussi la transformation TR_AEE et nous nous retrouvons dans la même situation que pour ❸ à ❺.

L'ensemble MS(S3,S2) contiendra exactement les correspondances introduites sauf pour les correspondances

```
CS : EA PROFESSEUR*Equivalent*Professeur.COURS
CS : EA CYCLE*Equivalent*Nom_cycle.NIVEAU
```

qui deviennent:

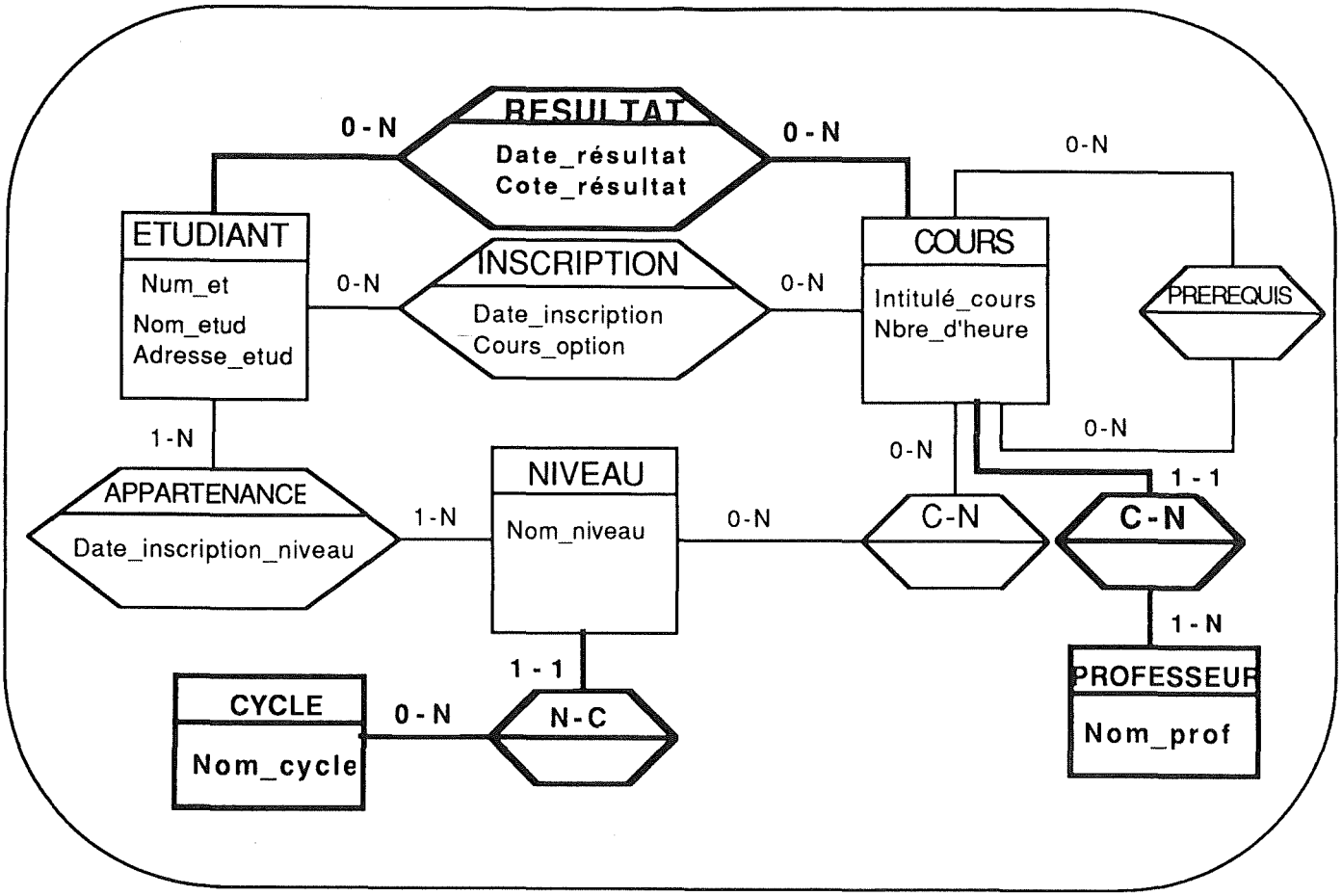
```
CS : EE PROFESSEUR*Equivalent*PROFESSEUR
CS : EE CYCLE*Equivalent*CYCLE.
```

Le résultat final de cette deuxième phase se trouve à la Figure 5–10. Les différences par rapport au résultat intermédiaire (voir Figure 5–9) sont mises en gras.

5.8 Noyau minimal des fonctions d'intégration pour un atelier logiciel

Le noyau minimal est l'ensemble des fonctions d'intégration qu'un atelier logiciel doit impérativement posséder dans sa boîte à outils pour pouvoir réaliser un processus d'intégration. Les fonctions d'intégration qui ne s'y trouvent pas peuvent se ramener à une des fonctions de ce noyau par l'application de certaines transformations sur les schémas conceptuels.

Figure 5–10: Résultat final des deux phases d'Intégration



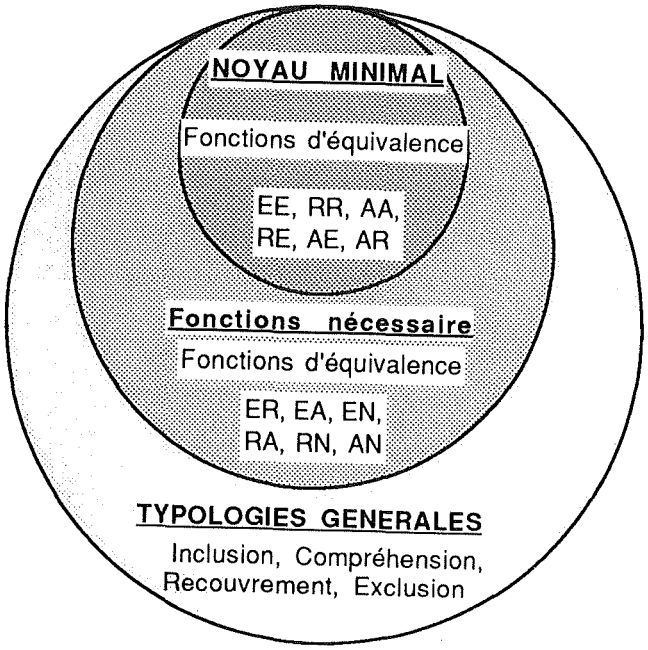
La Figure 5–11 montre une découpe possible de l'ensemble des fonctions d'intégration en trois niveaux.

Le niveau de base est le noyau minimal. Il comprend les fonctions indispensables à un processus d'intégration. Ce noyau est composé de six fonctions basées sur la typologie de l'équivalence.

Ces fonctions indispensables sont:

- La **fonction EE** d'intégration d'une entité du schéma secondaire avec une entité dans le schéma primaire,
- la **fonction RR** d'intégration d'une association du schéma secondaire avec une association dans le schéma primaire,
- la **fonction RE** d'intégration d'une association du schéma secondaire avec une entité dans le schéma primaire,
- la **fonction AA** d'intégration d'un attribut du schéma secondaire avec un attribut dans le schéma primaire,

Figure 5–11: Noyau minimal



- la **fonction AE** d'intégration d'un attribut du schéma secondaire avec une entité dans le schéma primaire, et
- la **fonction AR** d'intégration d'un attribut du schéma secondaire avec une association dans le schéma primaire.

Le deuxième niveau, les fonctions nécessaires, contient en plus des fonctions du noyau minimal les autres fonctions basées sur la typologie de l'équivalence énoncées à la section 5.5.
Ces fonctions sont "facultatives" car nous pouvons implémenter ces fonctions en utilisant les fonctions du noyau minimal de la façon suivante:

- la **fonction ER**, en transformant l'association en entité, nous nous ramenons à la fonction EE,
- la **fonction EA**, en transformant l'attribut en entité, nous nous ramenons à la fonction EE,
- la **fonction EN**, en créant une entité dans le schéma primaire, nous nous ramenons à la fonction EE,
- la **fonction RA**, en transformant l'attribut en entité, nous nous ramenons à la fonction RE,
- la **fonction RN**, en créant une association dans le schéma primaire, nous nous ramenons à la fonction RR, et
- la **fonction AN**, en créant un attribut dans le schéma primaire, nous nous ramenons à la fonction AA.

Le troisième niveau contient les fonctions permettant de traiter les typologies générales (i.e. l'inclusion, la compréhension, le recouvrement et l'exclusion décrites à la section 5.4.2). Nous avons vu lors de la section 5.5 comment ramener toutes ces typologies à la typologie d'équivalence.

Nous pouvons donc dire que le concepteur d'un outil d'intégration peut se résoudre à n'implémenter que les fonctions du noyau minimal pour réaliser les fonctions énoncées à la section 5.5. Les autres fonctions ne sont qu'une simple application des fonctions de ce noyau, avec au préalable quelques transformations sur les objets du schéma primaire.

CHAPITRE 6

IMPLEMENTATION DES FONCTIONS NECESSAIRES A UN ATELIER LOGICIEL: TRAMIS

6.1 Introduction

Après la définition de la méthode générale d'intégration de schémas conceptuels (voir Chapitre 5), nous abordons l'atelier logiciel qui va servir de support à notre méthodologie.

Nous nous limiterons dans la conception d'un module d'intégration qui implémente les différentes fonctions nécessaires d'intégration pour un contexte d'atelier logiciel. Nous donnerons la description détaillée des différentes fonctions du module d'intégration.

Pour terminer ce chapitre, nous citerons les limites de notre module d'intégration par rapport à la méthodologie générale développée au Chapitre 5.

6.2 L'atelier logiciel existant

Nous présentons successivement l'environnement WINDOWS dans lequel l'atelier logiciel TRAMIS a été conçu puis les possibilités actuelles de cet atelier que nous nous proposons d'étendre à l'intégration de spécifications de schémas.

6.2.1 Présentation de l'environnement *MS-WINDOWS*

Cet environnement est un produit de la firme MicroSoft (d'où le préfixe MS). Grâce à lui, un utilisateur peut simuler le multitâche sur son ordinateur de type Personnal Computer (PC) ou compatible. L'environnement permet de charger plusieurs applications en mémoire par le fait que WINDOWS offre pour chaque application une fenêtre pour en permettre l'exécution: en choisissant les fenêtres à ouvrir, l'utilisateur décide quelles applications exécuter. La description de WINDOWS que nous vous présentons est extraite de *[Frantz 90]*.

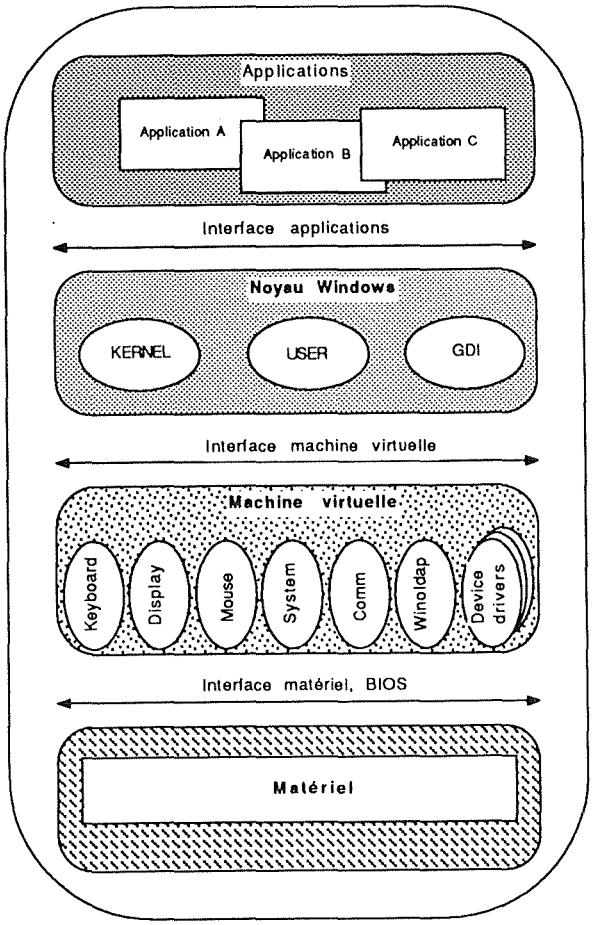
"WINDOWS n'est pas à proprement parler un système d'exploitation, bien qu'il en offre certaines fonctions. On utilise parfois le terme d'environnement d'exploitation qui insiste sur l'interface d'utilisation mais WINDOWS est bien plus qu'un moyen d'afficher et de saisir des données."

6.2.1.1 Contenu de MS-WINDOWS

"WINDOWS a été conçu pour éviter au programmeur de se soucier du type de matériel sur lequel tourne son application. Tous les périphériques habituels d'un micro-ordinateur sont *virtualisés*: clavier, souris, écran, imprimante, table traçante, moyen de communication asynchrone offrent une interface standard, parfaitement spécifiée et complète, totalement indépendante du matériel en place."

"WINDOWS est constitué de deux ensembles fonctionnels: le noyau WINDOWS, indépendant de tout matériel et une machine virtuelle réalisant l'interface avec le matériel (voir Figure 6–1)."

Figure 6–1: Les ensembles fonctionnels de WINDOWS



Le noyau

"Le noyau WINDOWS est un ensemble de logiciels, fournis par MicroSoft, et constituant le moteur, ou le coeur du système WINDOWS, indépendant du matériel. Le noyau est donc le même, quelle que soit la machine sur laquelle il est installé."

"Il comprend trois modules:

- **KERNEL**, le noyau proprement dit, est constitué des modules chargés du contrôle et de l'allocation des ressources nécessaires à WINDOWS. Ainsi, c'est KERNEL qui gère la mémoire du système (allocation de mémoire), charge les applications (chargeur) et réalise le *scheduling*, c'est-à-dire le multitâche.

- **USER** comprend le gestionnaire de fenêtres (*WINDOWS manager*), chargé de créer et maintenir les fenêtres à l'écran. C'est ce module qui déplace les fenêtres, modifie la taille, gère la superposition, les icônes[1] et curseurs .... USER reçoit également les informations des moyens d'entrée (clavier, souris, etc.) et les diffuse aux fenêtres destinataires.
- **GDI** (*Graphics Device Interface*) contient la bibliothèque graphique et permet de créer des images sur un périphérique (*device*) de sortie (écran, imprimante ...)."

"Pour mettre en oeuvre les fonctions qui font appel au matériel, le noyau se repose sur l'interface machine, qui lui présente les services de la machine virtuelle."

La machine virtuelle

"La machine virtuelle est constituée d'un ensemble de pilotes (*drivers*) gérant les ressources matérielles dépendantes du système physique."

"Chaque driver offre un ensemble de services parfaitement spécifiés au noyau. Il les implémente de façon différente selon le matériel." En consultant le schéma Figure 6–1, vous verrez quelques exemples de ces drivers dont le nom indique clairement ce qu'ils pilotent.

"D'autres modules entrent dans la composition de la machine virtuelle, les pilotes de périphériques (*Device Drivers*). Leur rôle est de reproduire physiquement sur un périphérique de sortie des dessins ou des textes à partir d'une interface de programmation unique. Chaque pilote de périphérique gère un périphérique spécifique. Ainsi, il existe un pilote de périphérique pour chaque type d'imprimante supportée par WINDOWS."

6.2.1.2 La place de WINDOWS

"Cet éclairage sur la constitution de WINDOWS permet de mieux comprendre sa place dans un PC. Une application développée pour WINDOWS[2] utilise les services de WINDOWS pour les périphériques (écran, souris, imprimantes, tables traçantes, communication asynchrones, timer) ainsi que pour la gestion de la mémoire. Mais elle doit utiliser le système DOS[3] pour accéder aux disques, aux répertoires, aux fichiers" comme on peut le voir sur le schéma ci-dessous.

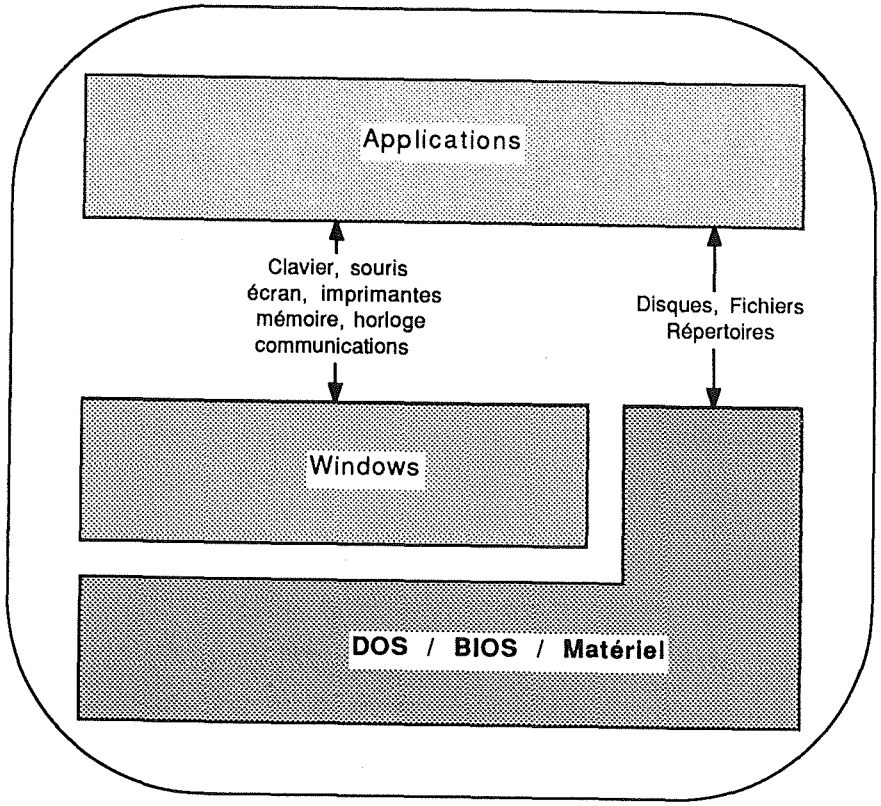
---

[1] Les icônes représentent les applications chargées en mémoire et inactives (i.e. qui n'ont pas de fenêtre ouverte).

[2] Comme celle que nous avons réalisée pour intégrer des schémas de bases de données conceptuelles décrites avec le modèle Entité/Association.

[3] Le système d'exploitation du disque (Disk Operating System) du PC.

Figure 6–2: Place de WINDOWS dans le PC



6.2.1.3 Qu'apporte WINDOWS ?

"WINDOWS apporte à l'utilisateur un certain nombre de possibilités inconnues sous DOS:

Le multitâche. Même s'il n'est pas préemptif, il permet de faire fonctionner simultanément plusieurs applications sur une même machine. WINDOWS se charge de partager l'écran, la mémoire, le clavier, la souris, le timer (horloge). Dans un système multitâche non préemptif, un programme ne peut être interrompu par le système d'exploitation. Il doit, de lui-même, redonner la main au système, pour lui permettre d'appeler les autres programmes.

L'indépendance vis-à-vis du matériel, ce qui garantit une meilleure portabilité d'une machine à une autre.

La communication entre applications, grâce à divers moyens: messages, partage de mémoire, presse-papiers, ...

L'édition de liens dynamiques, notion inconnue sous DOS, mais qui est la règle sous WINDOWS: le code commun à plusieurs applications peut être partagé, donc chargé une seule fois. De plus l'édition des liens (*link*) ne se fait qu'au moment de l'exécution."

6.2.2 Présentation de l'environnement *TRAMIS*[4]

TRAMIS est un environnement professionnel et pédagogique de conception de systèmes d'information conçu pour modéliser les données et les traitements. La gamme des outils TRAMIS[5] offre des outils de spécification graphiques, spécialisés surtout dans la construction des structures des bases de données.

On sous-estime le travail de conception de bases de données, travail souvent dévolu à des informaticiens. Dès lors, TRAMIS peut aider les personnes dans ce travail en permettant d'obtenir une représentation formelle la plus fidèle possible à la réalité. TRAMIS a réussi à concilier deux objectifs contradictoires de la conception d'une base de données, à savoir une représentation formelle fidèle et un serveur de données efficace. De plus, comme TRAMIS est pédagogique, il permet à des utilisateurs novices de concevoir des schémas.

Architecture de TRAMIS

TRAMIS est organisé comme une collection extensible d'outils travaillant sur une base de spécifications commune. Les outils reprennent un navigateur dans la base de données, des transformateurs, des vérificateurs de corrections, des générateurs de rapport ou de schémas physiques (DDL), . . . (voir Figure 6–3). Cet ensemble d'outils travaillent dans le contexte de MS-WINDOWS qui offre aux outils l'interface pour gérer les dialogues et coordonner les différents outils.

Le modèle unique de TRAMIS

Développer une base de données est un processus complexe généralement décomposé en plusieurs sous-processus séquentiels: cela commence par l'analyse des besoins et se termine par la génération du schéma physique (DDL). En général, chaque étape nécessite un modèle de description propre. TRAMIS par contre, offre un modèle unique pour tous les sous-processus pour qu'un concepteur puisse être aidé tout au long de la création de sa base de données.

Le modèle adopté est le modèle Entité/Association étendu qui supporte certains aspects:

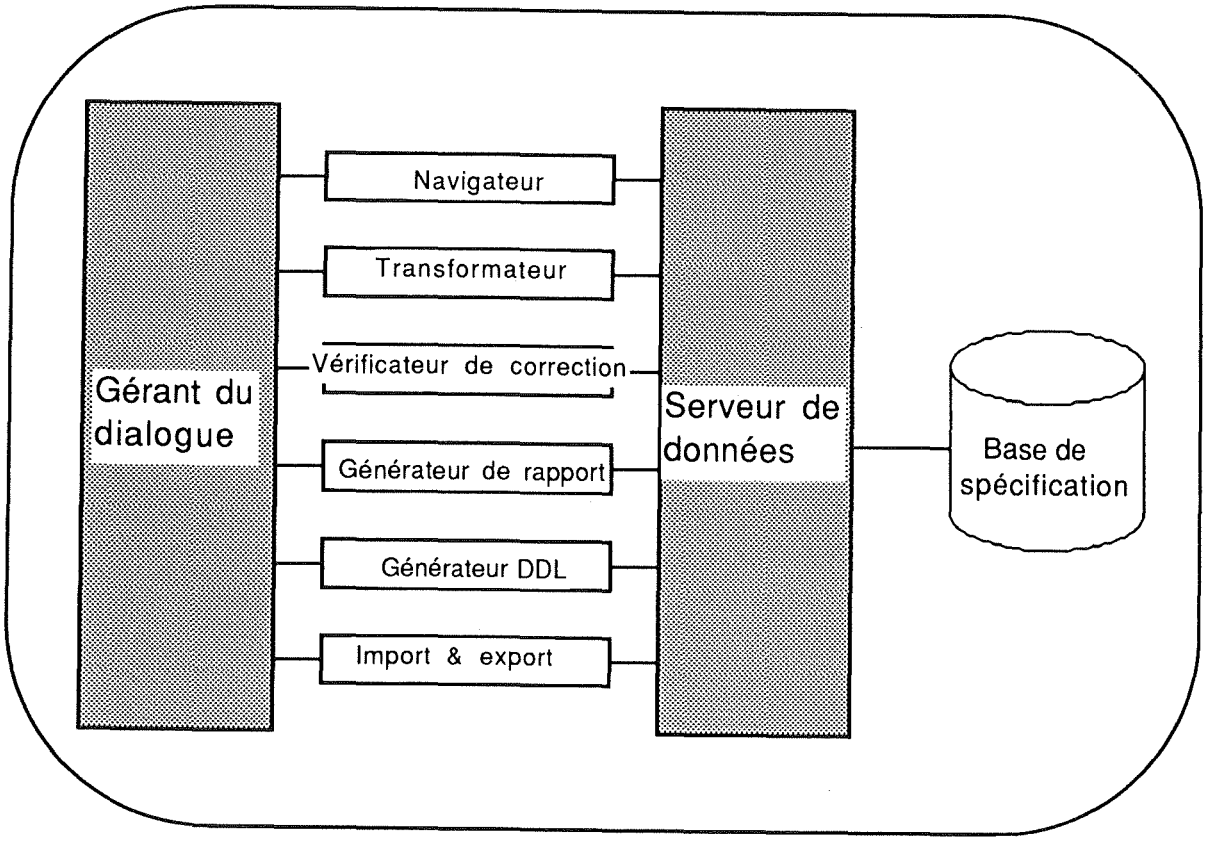
- la facette de désignation des objets au moyen d'un nom naturel, d'un nom court et d'un nom technique,
- la facette structurelle établit des liens entre les entités, les associations, . . . , notamment le groupement de certains objets,
- la facette statistique qui enrichit le schéma conceptuel pour réaliser des optimisations lors du passage au schéma physique,
- la facette de description des objets au moyen de textes informels qui complète le schéma. Cette description est utilisée dans la génération des rapports.

---

[4] Cette présentation est basée sur les articles *[Hainaut 90]* et *[Brès 90]*

[5] L'ensemble des produits TRAMIS, développés conjointement par la firme CONCIS (France) et les Facultés Notre-Dame de la Paix (Namur) comprend les programmes "TRAMIS/VIEW", "TRAMIS/FLOW" et "TRAMIS/MASTER". Dorénavant, lorsque nous parlerons de TRAMIS, nous sous-entendrons "TRAMIS/MASTER".

Figure 6–3: L'architecture de TRAMIS



L'approche transformationnelle de TRAMIS

Une transformation de schéma est une opération dans laquelle une structure est remplacée par une autre qui lui est équivalente. Une transformation sert par exemple à raffiner/généraliser un concept, intégrer des schémas partiels, produire un schéma conforme à un modèle de représentation des données, ... TRAMIS fournit un ensemble de transformations qui peut être utilisé à n'importe quel niveau de la conception.

La notion d'équivalence de schéma est importante: elle garantit au concepteur que les transformations effectuées par l'outil (TRAMIS) conserveront la sémantique qu'il a voulu modéliser dans son schéma[6].

Les transformations de TRAMIS préservent donc la sémantique et garantissent en plus la conservation des structures techniques, des noms et des statistiques. Chaque transformation fait appel à l'outil de validation qui fait partie de TRAMIS pour vérifier que les conditions à remplir sont bien respectées.
TRAMIS propose des transformations à trois niveaux:

- les **transformations élémentaires**: une modification est apportée à un objet et le concepteur peut en suivre pas à pas l'évolution car certaines transformations peuvent

[6] Si l'équivalence préserve la sémantique d'un schéma, rien ne dit qu'on la retrouve dans le format initial: c'est justement parce que ce format était incompatible avec une certaine syntaxe (par exemple, la façon de représenter les choses dans le modèle relationnel, qui ne contient pas le concept d'association).

être résolues de différentes façons; ainsi, il pilote le changement pour obtenir le résultat qu'il souhaite,

- les **transformations globales**: une modification est appliquée à tous les objets vérifiant un critère donné par le concepteur, toujours selon la même philosophie de suivre à la trace le processus,
- les **transformations imposées par un modèle**: toutes les structures non compatibles avec le modèle qui sera utilisé pour le schéma physique sont remplacées automatiquement par une représentation équivalente et compatible; le concepteur peut par la suite y apporter certains raffinements (pour améliorer l'efficacité, . . .).

L'approche boîte à outils de TRAMIS

Les différents outils de TRAMIS sont indépendants les uns des autres et peuvent être utilisés librement. Ils utilisent la même interface (celle de MS-WINDOWS) ce qui rend leur utilisation facile, pour les experts comme pour les novices.

C'est le concepteur qui choisit l'outil dont il a besoin et MS-WINDOWS qui en dirige l'exécution. Les outils peuvent coopérer grâce aux informations qui sont stockées dans la base de spécifications gérée par un serveur[7] (qui fait partie des outils de TRAMIS).

A tout moment de la conception, l'utilisateur peut donc appeler l'outil qui va vérifier la conformité du schéma à l'instant de l'appel avec un des modèles connu par TRAMIS. Le programme vérifiera si les informations contenues dans le schéma à analyser satisfont les contraintes imposées par le modèle choisi par l'utilisateur.

6.3 Conception du module d'intégration

Nous décrivons au préalable la partie du schéma SGBD de TRAMIS qui concerne notre module d'intégration. Après cela, nous verrons la dynamique de notre algorithme. Nous donnerons la description des différentes fonctions réalisées dans le pseudo-code ADL (décrit dans *[Hainaut 86a]*, aux chapitres V et VI). Pour conclure, nous énoncerons les quelques restrictions de notre module d'intégration par rapport à la méthodologie que nous avons définie lors du Chapitre 5.

6.3.1 Le méta-schéma de l'atelier TRAMIS/MASTER

Cette section décrit le sous-schéma de la base de données des spécifications selon les structures de données offertes par le SGBD de l'atelier TRAMIS. Cette description est basée sur un sous-ensemble du MAG[8]. Ce modèle décrit les structures de données non seulement sous l'angle de la sémantique qu'expriment ces données, mais aussi sous celui des accès dont elles peuvent faire l'objet.

Les principales restrictions du MAG par rapport au modèle Entité/Association de *[Chen 76]* sont les suivantes:

1. les types d'associations sont de connectivité 1-N ou N-1,
2. un type d'associations ne peut avoir d'attribut.

Nous donnons ci-après la description du schéma SGBD de l'atelier (voir Annexe A)

---

[7] Ce serveur n'est pas accessible par l'utilisateur: il ne répond qu'aux demandes des outils en leur fournissant les informations dont ils ont besoin pour accomplir pleinement leur fonction.

[8] MAG: Modèle d'Accès Généralisé. Le lecteur pourra consulter le chapitre V du livre de *[Hainaut 86a]*.

Description sémantique du schéma SGBD de l'atelier TRAMIS/MASTER

Nous ne décrirons que la partie du schéma de la base de données des spécifications qui nous intéresse pour l'élaboration du module d'intégration dans l'atelier. Le lecteur intéressé par le schéma complet pourra consulter *[Hainaut 86b]*.

SCHEMA Un base de données peut avoir un ou plusieurs **SCHEMA**. Chaque SCHEMA est identifié par son nom (**SCH-NAME**).

ENTITY-T L'élément de base d'un SCHEMA est le type d'entités (**ENTITY_T**). Tout ENTITY-T appartient à un et un seul SCHEMA (via **E-IN**).

L'ENTITY-T décrit les notions traditionnelles de Type d'Entités, Schéma de Relation, Type d'Articles, de Segments, d'Enregistrements, . . . , que l'on trouve dans toute description de données structurées. Les ENTITY-T d'un SCHEMA sont identifiés par leur nom (**E-NAME**). Elles peuvent avoir un nom court (**SHORT-NAME**) et une date de la dernière mise-à-jour (**DATE**).

LINK Le type d'associations entre types d'entités est représenté par un **LINK**, qui relie au moins deux ENTITY-T (non nécessairement distincts). Les LINK d'un SCHEMA sont identifiés par leur nom (**L-NAME**). Un LINK est caractérisé tout comme un ENTITY-T par un nom court (**SHORT-NAME**) et une date de la dernière mise-à-jour (**DATE**). Tout LINK appartient à un et un seul schéma (via **L-IN**).

ROLE Tout ENTITY-T participant à un LINK doit jouer un rôle (**ROLE**). Un ROLE est caractérisé par un nom (**R-NAME**), un nom court (**SHORT-NAME**). Un même ROLE d'un LINK peut être joué par un ou plusieurs ENTITY-T (cela est représenté par **E-ROLE**). En plus de cela, on peut indiquer le nombre minimum (**MIN-CON**) et le nombre maximum (**MAX-CON**) de fois qu'une même occurrence d'une ENTITY-T peut apparaître dans un LINK via un ROLE. Les valeurs les plus fréquentes de MIN-CON SONT 0 et 1, tandis que celles de MAX-CON sont 1 et N[9].

ATTRIBUTE La propriété qui caractérise un ENTITY-T ou un LINK est représentée par le concept d'attribut (**ATTRIBUTE**) possédant un nom (**AT-NAME**) et qui lui est associé via **ATT-OF**. Un attribut peut avoir aussi un nom court (**SHORT-NAME**), une date de dernière mise-à-jour (**DATE**). Un ATTRIBUTE peut être simple ou décomposable en ATTRIBUTES plus élémentaires via ATT-OF. L'ATTRIBUTE est aussi caractérisé par **MIN-REP** et **MAX-REP** qui représentent respectivement le nombre de valeurs minimum et maximum de l'ATTRIBUTE qu'on retrouve associé à un ENTITY-T, LINK ou ATTRIBUTE décomposable.

RELATION Une association entre deux objets quelconques se représente par un objet **RELATION** attaché à ces deux objets via **MEMBER1** et **MEMBER2**. RELATION nous permet dans le cas de notre module d'intégration de mémoriser les correspondances entre les objets du schéma secondaire et les objets du schéma primaire. Pour faire la distinction entre les autres RELATION possibles dans l'atelier, **TYPE** aura la valeur "INT". La RELATION est en outre caractérisée par **TYPE-O1** et **TYPE-O2** qui représentent respectivement le type de l'objet mis en correspondance dans le schéma secondaire et le type de l'objet mis en correspondance dans le schéma primaire. TYPE-O1 et TYPE-O2, dans le cas de notre module d'intégration ne peuvent prendre qu'une des trois valeurs suivantes:

— "A" pour Attribut;

— "E" pour Entité;

[9] Ce nombre est représenté dans l'atelier TRAMIS par la valeur conventionnelle 99999.

— "R" pour Relation (association).

6.3.2 Dynamique générale de l'ensemble des modules

Nous présentons ici l'enchaînement dynamique des différentes actions à réaliser pendant le processus d'intégration, à partir du moment où l'utilisateur à déclaré qu'il voulait intégrer des schémas dans la base de données active.
Cet algorithme est le même que celui que nous avons décrit dans la section 5.6 mais il est spécialisé au module (et aux fonctions) que nous avons implémenté.

Nous utiliserons pour schématiser cet algorithme, un formalisme semblable à celui de la dynamique des flux que nous retrouvons chez *[Bodart & Pigneur 89]*. Nous donnons la description des différents concepts, illustrés à la Figure 6–4.

- Un *traitement* correspond à une procédure qui accomplit une fonction élémentaire telle que l'affichage à l'écran du schéma sélectionné par l'utilisateur.
- Un *ensemble de traitements à détailler* correspond à un ensemble de fonctions. L'ensemble est explosé en une vue plus restreinte que nous retrouvons aux Figure 6–6 et Figure 6–7.
- Un *message de l'utilisateur* est un renseignement fourni interactivement par l'utilisateur pour que le processus puisse se poursuivre normalement.
- Un *test* correspond à l'analyse d'un résultat interne (e.g. du traitement qui le précède). Dans la plupart des cas, il s'agit d'un test *SI OUI, emprunter le chemin "OUI", SINON, parcourir le chemin "NON"*. Seul le test "T2" (voir Figure 6–5) est un peu différent.
- Une *fin de l'algorithme* correspond à l'achèvement du processus.
- Un *début et fin d'un détail de traitement* est employé pour marquer le début et la fin d'une explosion d'un ensemble de traitements. Ce symbole correspond au point de jointure entre le schéma principal et le schéma qui sert de "gros plan".

Ces différents concepts sont reliés par des flèches qui indiquent le parcours à suivre.

Voici la signification des noms que nous utilisons dans la Figure 6–5:

— Ss et Sp désignent le Schéma secondaire et le Schéma primaire,

— Os et Op représentent respectivement un Objet de Ss et de Sp,

— Ts et Tp indiquent respectivement le type de ces objets dans Ss et dans Sp, à savoir "E" pour Entité, "R" pour Relation et "A" pour Attribut,

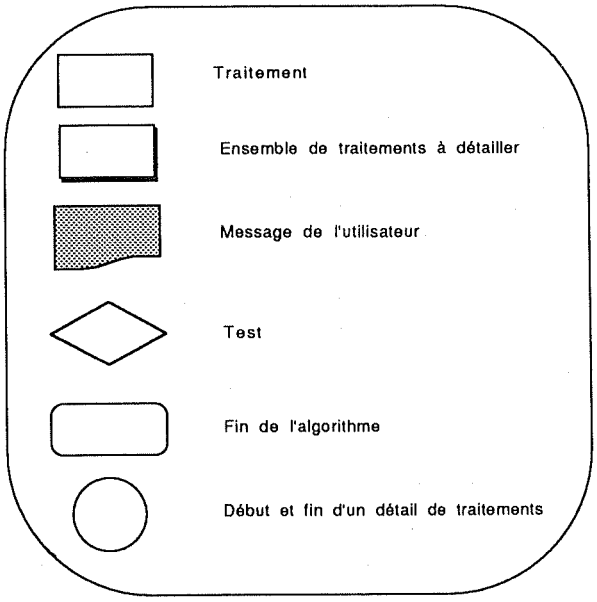
— les fonctions telles que "Integ. EA" sont celles spécifiées au Chapitre 5,

— le message "menu" correspond au choix présenté à l'utilisateur par l'outil d'intégration de passer, soit à l'intégration, soit de sélectionner le menu **sortir** pour terminer le processus ou soit de sélectionner le menu **correspondances** pour voir la liste des correspondances déjà introduites,

— enfin les tests "Ti" ont les significations suivantes:

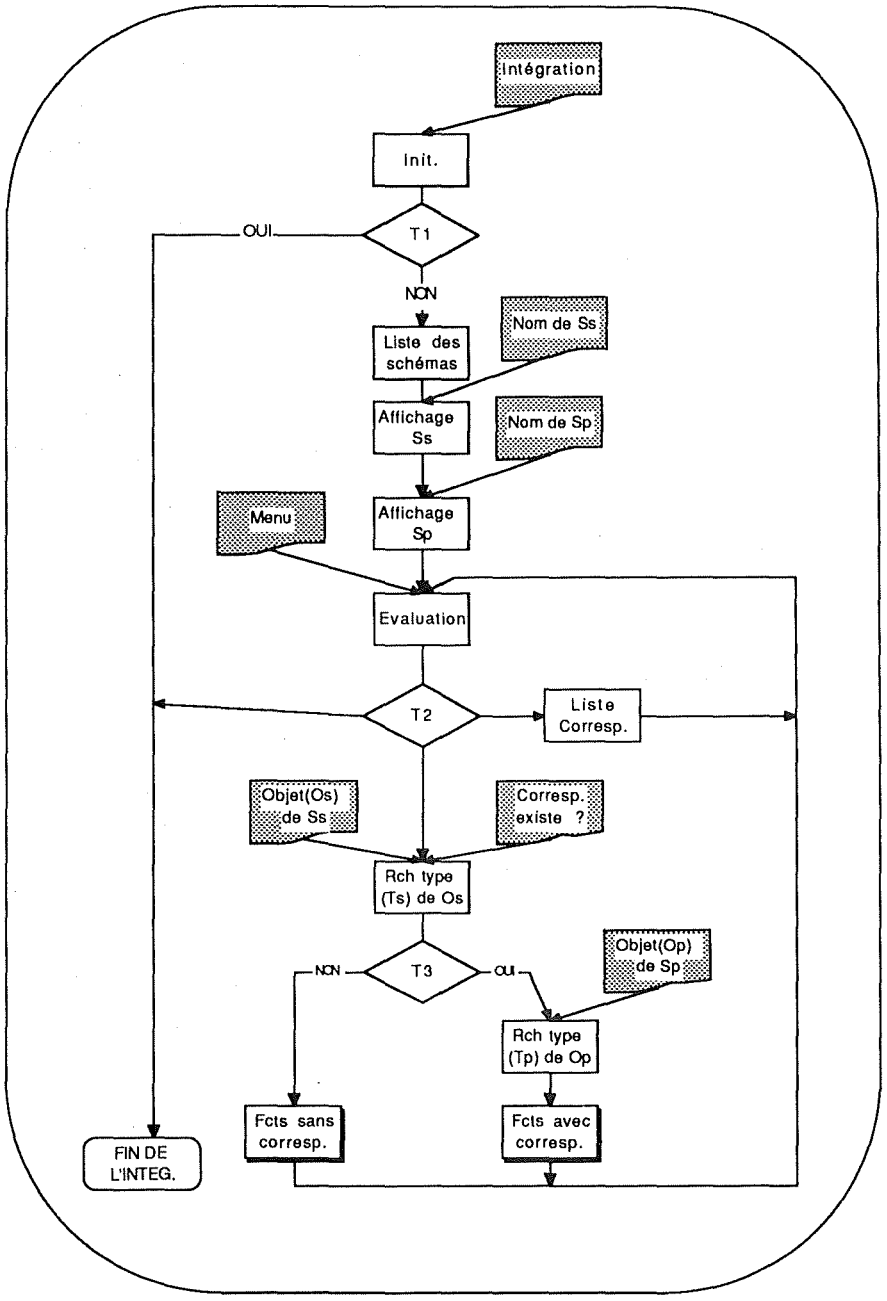
1. La base de données choisie contient-elle un seul schéma ?

Figure 6–4: Les concepts utilisés



2. Si le menu choisi est **sortir**, prendre la branche de gauche (voir Figure 6–5) et terminer,
   si le menu choisi est **correspondances**, prendre la branche de droite et afficher les correspondances,
   sinon, aller tout droit et commencer à intégrer
3. Existe-t-il un correspondant à Os ?
4. Ts = 'E' et Tp = 'E' ? (voir Figure 6–6)
5. Ts = 'E' et Tp = 'R' ?
6. Ts = 'E' et Tp = 'A' ?
7. Ts = 'R' et Tp = 'R' ?
8. Ts = 'R' et Tp = 'E' ?
9. Ts = 'R' et Tp = 'A' ?
10. Ts = 'A' et Tp = 'A' ?
11. Ts = 'A' et Tp = 'E' ?

Figure 6–5: Dynamique de l'algorithme



6.3.3 Description des traitements de l'algorithme d'intégration

Init est appelé quand un ordre d'intégration est donné par l'utilisateur. Ce traitement vérifie que la base de données ouverte contient au moins deux schémas. Si c'est le cas, il efface toutes les correspondances que contient la base de données. Cela permet de reprendre un schéma temporaire et de l'intégrer avec un nouveau schéma car notre méthodologie interdit que chaque objet ne soit mis en correspondance plus d'une fois (voir Chapitre 5).

Figure 6–6: Détail des fonctions avec correspondances

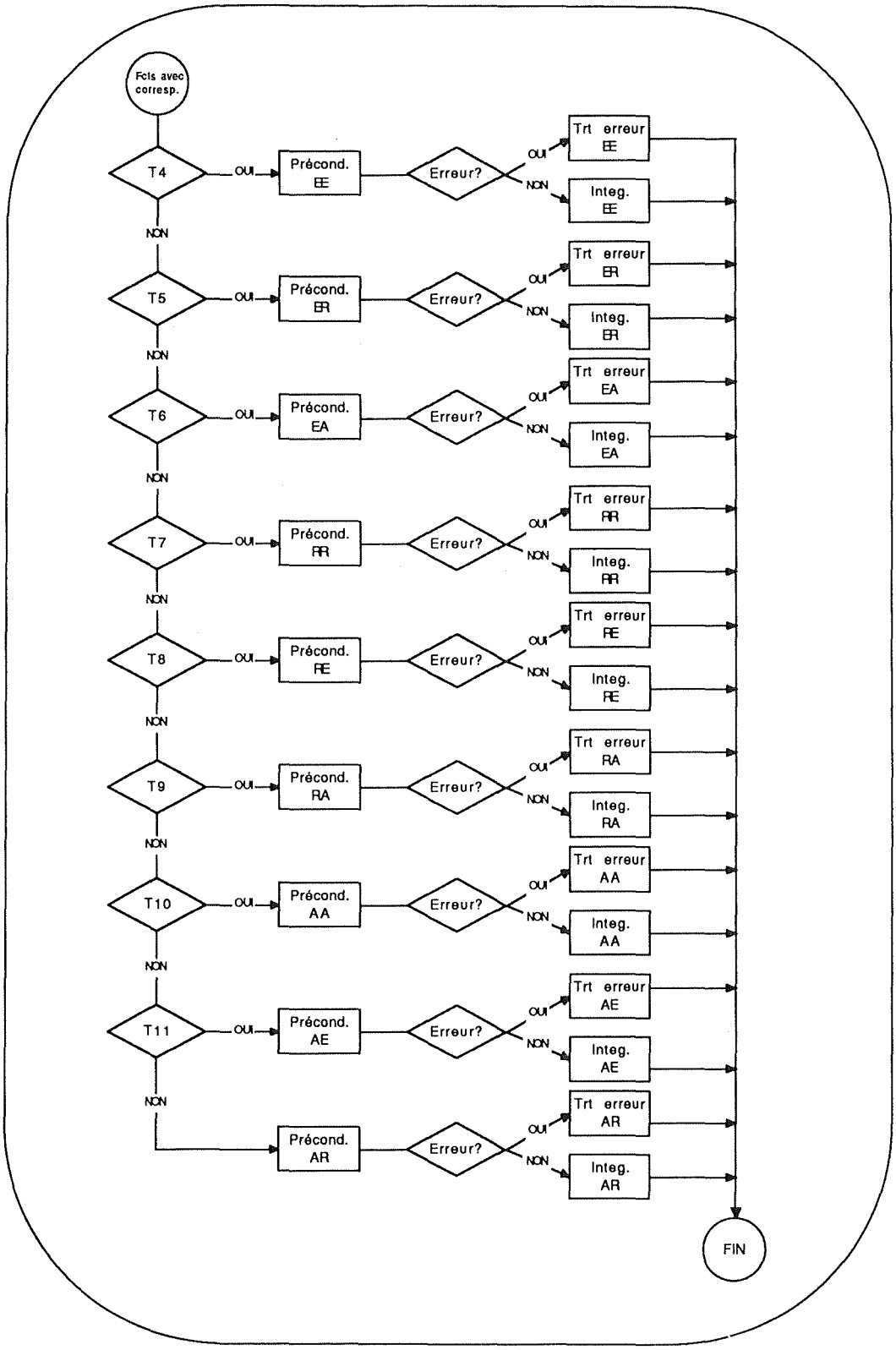
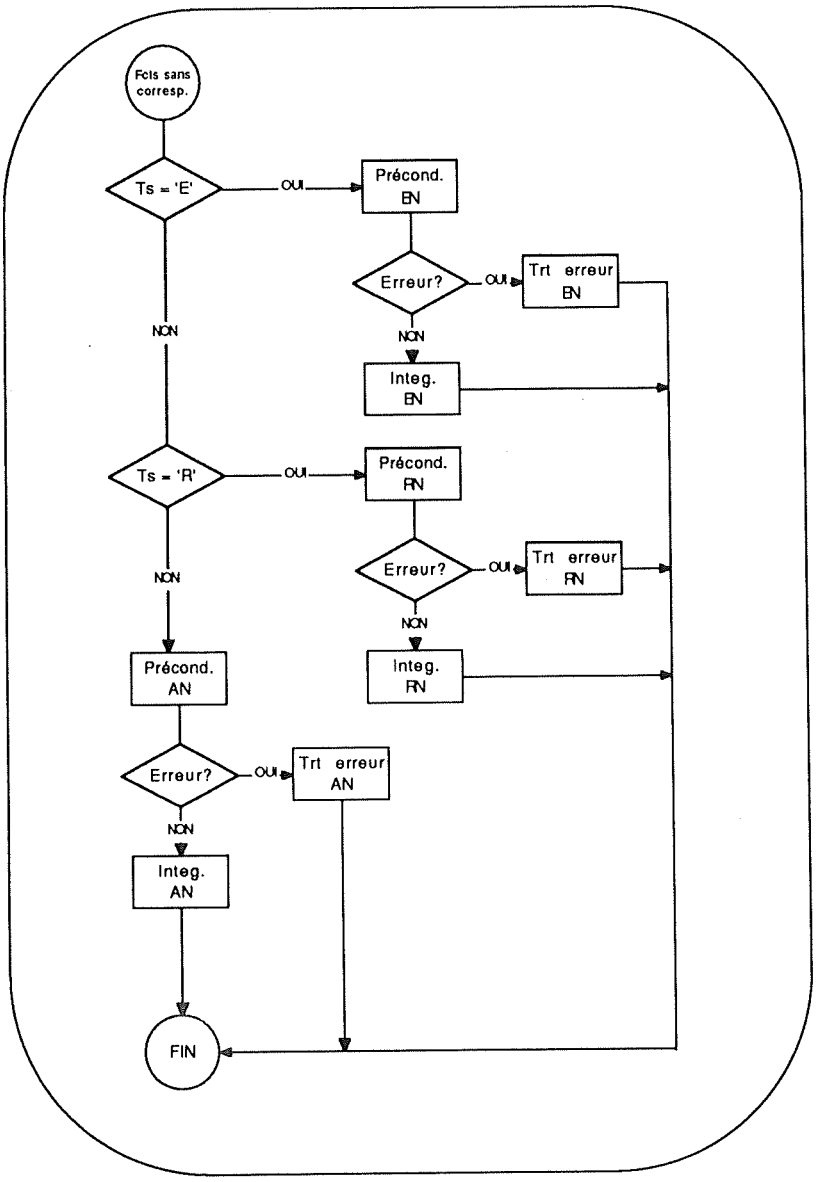


Figure 6–7: Détail des fonctions sans correspondance



Liste des schémas affiche la liste des schémas disponibles pour l'intégration dans la base de données ouverte.

L'utilisateur choisit son schéma secondaire et **Affichage Ss** affiche son contenu à l'écran.

L'utilisateur choisit le schéma primaire parmi les schémas restants et **Affichage Sp** affiche son contenu à l'écran.

Evaluation attend que l'utilisateur produise un évènement en choisissant soit le menu "Correspondances", soit le menu "Sortir" ou bien sélectionne un objet à intégrer dans le schéma secondaire.
Une fois cet évènement détecté, le test **T2** le dirige vers le traitement qui lui est associé.

Liste corresp. affiche l'ensemble des correspondances déjà introduites dans la base de données: l'utilisateur à le choix de voir ces correspondances du schéma primaire vers le schéma secondaire ou vice versa.

Rch type (Ts) de OS analyse le type de l'objet du schéma secondaire et demande à l'utilisateur si cet objet (Os) a un correspondant.

S'il n'a pas de correspondant, l'algorithme vérifie les préconditions propres au type de Os grâce à **Précond. ?N**[10] (voir Figure 6–7).
Si les préconditions ne sont pas remplies, **Trt erreur ?N** indique la cause du refus par un message d'erreur. Sinon **Integ ?N** réalise l'intégration.

Si un correspondant existe, l'utilisateur doit choisir cet objet (Op) dans le schéma primaire et **Rch type (Tp) de Op** en analyse le type.

L'algorithme vérifie ensuite les préconditions associées aux types de Os et Op grâce à **Précond. ??**[11] (voir Figure 6–6.
Si les préconditions ne sont pas remplies, **Trt erreur ??** indique la cause du refus par un message d'erreur. Sinon **Integ ??** réalise l'intégration.

Dans les deux cas, lorsque l'intégration est terminée, l'algorithme revient au traitement **Evaluation** et attend la commande suivante de l'utilisateur.

6.3.4 Méta-algorithme ADL des fonctions d'intégration

Nous donnons uniquement à titre d'exemple les algorithmes concernant la fonction d'intégration entre une entité et un attribut (fonction EA). Cette fonction est assez complexe pour donner un aperçu général des différentes fonctions composant notre module d'intégration. Le méta-code de celui-ci est fourni en annexe (voir Annexe B).

La fonction EA comprend trois algorithmes:

— **EA_aff_err**: l'algorithme qui affiche à l'écran les éventuelles erreurs,

— **EA_integ**: l'algorithme d'intégration de la correspondance EA,

— **EA_precond**: l'algorithme qui vérifie les préconditions de la fonction EA.

Fonction: EA_aff_err

```
procédure EA_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance EA.

Précondition:

     TypeObjectInt1 = 'E', ❶
     TypeObjectInt2 = 'A', ❷
     mess_err : numéro du message d'erreur (compris entre 1 et 3).

Postcondition:
```

[10] Le symbole "?" représente n'importe quel caractère parmi "E", "R" ou "A" correspondant au type de l'objet Os sélectionné dans le schéma secondaire.

[11] Le symbole "?" représente n'importe quel caractère parmi "E", "R" ou "A" correspondant respectivement au type de l'objet Os et au type de l'objet Op.

```
     Si mess_err = 1, le message "L'entité sélectionnée dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "L'attribut sélectionné dans
                      le schéma primaire a déjà été mis en
                      correspondance" est présenté à l'utilisateur;
        mess_err = 3, le message "L'attribut sélectionné dans le
                      schéma primaire n'est pas de niveau 1"
}
begin
 if mess_err = 1
 then
  display "L'entité sélectionnée dans le schéma secondaire a déjà
           un correspondant"; ❸
 else
   if mess_err = 2
      display "L'attribut sélectionné dans le schéma primaire a déjà
               été mis en correspondance";
   else
      display "L'attribut sélectionné dans le schéma primaire n'est
               pas de niveau 1";
   endif;
 endif;
end;
```

❶ C'est une variable globale qui contient le type de l'objet du schéma secondaire que l'utilisateur désire intégrer.

❷ C'est une variable globale qui contient le type de l'objet du schéma primaire.

❸ Cette commande permet de présenter à l'écran le message qui est mis entre guillemet

Fonction: EA_Integ

```
procedure EA_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section 5.5.2 pour l'intégration de
     l'entité sélectionnée dans le schéma secondaire avec
     l'attribut sélectionné dans le schéma primaire.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss), ❹
     CurSchName2 : nom du schéma primaire (Sp), ❺
     ObjectInt1  : nom de l'entité de Ss à intégrer dans Sp, ❻
     ObjectInt2  : nom de l'attribut de Sp à intégrer avec
                   ObjectInt1, ❼
     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'A',
     PereObjectInt2 : nom du père de ObjectInt2, ❽
     Niveau2     : niveau de ObjectInt2 dans la liste des attributs
                   de PereObjectInt2. ❾

Postcondition:

     MS(Ss,Sp)= MS + EE ObjectInt1*Equivalent*ObjectInt2.
}
   begin
      Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
      Ent:= ENTITY_T((E_S: Sch1)
                     and (:E_NAME = ObjectInt1));
      Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
      Type := Rch_type_pere(PereObjectInt2, Sch);
      if (Type = "E")
      then
        Pere:= ENTITY_T(: E_NAME = PereObjectInt2);
        Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
      else
        Pere:= LINK(: L_NAME = PereObjectInt2);
        Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
      endif;
```

```
if (Type = "R")
then
(10)    create EntLk:= ENTITY_T((: E_NAME = L_NAME(: Pere))
                            and (: SHORT_NAME = SHORT_NAME(: Pere)
                            and (: DATE = DATE(: Pere)
                            and (E_S: Sch2);

    Att:= first(ATTRIBUTE(AT_EL: Pere));
    while Att <> () do
       modify Att(At_EL: EntLk);
       Att:= next(ATTRIBUTE(AT_EL: Pere));
    endwhile;

    Rol:= first(ROLE(R_L: Pere));
    while Rol <> () do
     create Lk1:= LINK((L_S: Sch2)
                       and (: L_NAME = choisi par le système
                                       d'intégration)
                       and (: SHORT_NAME = les trois premiers
                                           caractères de L_NAME)
                       and (: DATE = date du système));
     create Rol1:= ROLE((: R_NAME = choisi par le système
                                    d'intégration)
                        and (: SHORT_NAME = les trois premiers
                                            caractères de R_NAME)
                        and (: DATE= date du système)
                        and (: MIN_CON = 1)
                        and (: MAX_CON = 1)
                        and (R_L: Lk1));
     create Erol:= E_ROLE((ER_E: EntLk)
                          and (ER_R: Rol1));
      Rol:= next(ROLE(R_L: Pere));
   endwhile;

   Rel:= RELATION((REL2_O: Pere)
                  (: TYPE = "INT"));
   if (Rel <> ())
   then
      modify Rel(REL2_O: EntLk);
   endif;

   delete Pere;

   Ent2:= EntLK;
else
   Ent2:= Pere;
endif;

(11) Create EntAtt:= ENTITY_T((: E_NAME = AT_NAME(: Att))
                          and (: SHORT_NAME = SHORT_NAME(: Att))
                          and (: DATE = DATE(: Att))
                          and (E_S: Sch2));

Create Att1:= ATTRIBUTE((: AT_NAME = At_NAME(: Att))
                        and (: SHORT_NAME = SHORT_NAME(: Att))
                        and (: DATE = DATE(: Att))
                        and (: MIN_REP = 1);
                        and (: MAX_REP = 1);
                        and (AT_EL: EntAtt));

Create Lk:= LINK ((: L_NAME = choisi par le système
                              d'intégration)
                  and (: SHORT_NAME = les trois premiers
                                      caractères de L_NAME)
                  and (: DATE = date du système)
                  and (L_S: Sch2));

Create Rol1:= ROLE ((: R_NAME = choisi par le système
                                d'intégration)
                    and (: SHORT_NAME = les trois premiers
                                        caractères de R_NAME)
                    and (: DATE = date du système)
                    and (: MIN_CON = MIN_REP(: Att))
                    and (: MAX_CON = MAX_REP(: Att))
                    and (R_L: Lk));

Create Erol1:= E_ROLE ((ER_R: Rol1)
                       and (ER_E: Ent2));
```

```
    Create Rol2:= ROLE ((: R_NAME = choisi par le système
                           d'intégration)
                        and (: SHORT_NAME = les trois premiers
                                            caractères de R_NAME)
                        and (: DATE = date du système)
                        and (: MIN_CON = 1)
                        and (: MAX_CON = MAX_REP(: Att))
                        and (R_L: Lk));

    Create Erol2:= E_ROLE ((ER_R: Rol2)
                           and (ER_E: EntAtt));
    delete Att;

    EE_integ; /* Intégration de Ent avec EntAtt */
end;
```

❹ C'est une variable globale qui contient le nom du schéma secondaire d'où proviennent les objets à intégrer.

❺ C'est une variable globale qui contient le nom du schéma primaire où l'utilisateur désire intégrer.

❻ C'est une variable globale qui contient le nom de l'objet du schéma secondaire que l'utilisateur désire intégrer.

❼ C'est une variable globale qui contient le nom de l'objet du schéma primaire.

❽ C'est une variable globale qui contient le nom du père de l'objet du schéma primaire.

❾ C'est une variable globale qui indique le niveau de l'attribut du schéma primaire.

❿ Nous transformons l'association en entité en suivant les étapes indiquées lors de l'explication de la transformation Tr_RE(Pere,EntLk) (voir Chapitre 5).

⓫ Nous transformons l'attribut en entité en suivant les étapes indiquées lors de l'explication de la transformation TR_AEE(Att, EntAtt).

Fonction: EA_precond

```
function EA_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'intégration d'une entité de Ss avec un attribut de Sp
     sont remplies. Ces conditions ont fait l'objet d'une
     description à la section 5.5.2.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'entité de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'attribut de Sp à intégrer avec ObjectInt1,
     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'A',
     PereObjectInt2 : nom du père de ObjectInt2,
     Niveau2     : niveau de ObjectInt2 dans la liste des attributs
                   de PereObjectInt2.

Postcondition:

     EA_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
                        mess_err = 2 si ObjectInt2 a déjà été mis
                                     en correspondance avec un objet
                                     de Ss;
                        mess_err = 3 si ObjectInt2 n'est pas un
                                     attribut de niveau 1.
}
begin
```

```
⑫  Sch:= SCHEMA(: SCH_NAME = CurSchName1);
    Ent:= ENTITY_T((E_S: Sch)
                   and (: E_NAME = ObjectInt1));
    if (RELATION((REL1_O: Ent)
                 and (: TYPE = "INT"))) <> 0
    then
        mess_err:= 1;
        return(FALSE);
    endif;

⑬  Sch:= SCHEMA(: SCH_NAME = CurSchName2);
    Type := Rch_type_pere(PereObjectInt2, Sch);
    if (Type = "E")
    then
      Pere:= ENTITY_T(: E_NAME = PereObjectInt2);
      Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
    else
      Pere:= LINK(: L_NAME = PereObjectInt2);
      Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
    endif;

    if (RELATION((REL2_O: Att)
                 and (: TYPE = "INT"))) <> 0
    then
        mess_err:= 2;
        return(FALSE);
    endif;

⑭  if (niveau2 <> 1)
    then
        mess_err:=3;
        return(FALSE);
    endif;

  return(TRUE); ⑮
end;
```

⑫ Cela teste la première condition, à savoir: si l'entité a déjà un correspondant dans le schéma primaire.

⑬ Cela teste la deuxième condition, à savoir: si l'attribut a déjà un correspondant dans le schéma secondaire.

⑭ Cela teste la troisième condition, à savoir: si l'attribut n'est pas de niveau 1.

⑮ Cette commande correspond au retour de la procédure avec le passage du résultat entre parenthèses.

6.3.5 Macros fonctions employées

Dans la section 6.3.4, nous avons déjà présenté deux de ces macros fonctions: "display" et "return".
Nous définissons maintenant les autres macros fonctions employée dans Annexe B:

- "count" sert à compter le nombre d'éléments dans un type d'entités du schéma SGBD de TRAMIS,
- "PAS_FIN_DE_L'INTEGRATION" correspond à l'évènement associé à la sélection par l'utilisateur du menu **sortir** (voir section 6.3.2),
- "select" correspond à la sélection par l'utilisateur d'un objet dans un schéma et à l'initialisation des variables globales avec l'objet sélectionné,
- "record" renvoie un objet qui est soit une entité, soit une association ou soit un attribut, et
- "accept" sert à demander à l'utilisateur une information et à la sauvgarder dans une variable.

6.3.6 Limitations de l'algorithme

Nous avons imposé quelques restrictions par rapport à la méthodologie générale pour obtenir un produit plus facilement réalisable. Les endroits où apparaissent ces limitations sont:

1. un attribut ne peut être transformé en entité que s'il est de niveau 1. Cela concerne les fonctions d'intégration EA et RA. Nous avons imposé que l'attribut soit de niveau 1 pour éviter de devoir appliquer une suite de transformations aux attributs se trouvant entre l'attribut que nous voulons intégrer et l'entité (ou l'association) à qui il appartient. Une solution pour contourner cette limitation est d'utiliser préalablement à l'intégration l'outil offert par l'atelier TRAMIS pour transformer des attributs décomposables en attributs simples (de niveau 1).
2. Les chemins examinés sont de longueur 1 uniquement. Cela signifie que pour les liens entre une entité et une association (par exemple pour l'intégration RR), nous n'examinons que les entités qui participent directement à l'association. Pour les liens entre deux entités (par exemple pour l'intégration RE), nous n'examinons que les entités séparées par au plus une association. Nous avons imposé ces limitations pour simplifier notre algorithme de recherche de chemins et pour éviter un nombre exponentiel de validations qu'imposeraient ces chemins plus longs.
3. Nous ne pouvons intégrer une association du schéma secondaire vers une association du schéma primaire que si cette dernière a un degré supérieur ou égal. Cette restriction est le fait de la définition de notre méthodologie qui impose comme précondition à l'intégration d'une association, du schéma secondaire dans le schéma primaire, que toutes les entités participant à celle-ci soient déjà mises en correspondance avec un objet du schéma primaire.
4. Nous autorisons d'intégrer des associations cycliques sans aucune autre validation que celles des préconditions. Normalement, on ne peut intégrer que deux associations cycliques entre elles.
5. Enfin, nous n'avons pas implémenté l'outil de recherche des homonymes parmi les objets à intégrer.

6.4 Conclusion

Nous avons implémenté un petit module d'intégration dans l'atelier TRAMIS. Celui-ci comprend l'ensemble des fonctions nécessaires qui réalise l'intégration sur base de l'équivalence entre les objets du schéma secondaire et les objets du schéma primaire.

Certaines de ces limitations peuvent être facilement éliminées (i.e. les limitations concernant le niveau des attributs). Les autres demande une analyse plus en profondeur qui pourrait être faite dans une recherche ultérieure.

Le module que nous avons réalisé est indépendant de l'interface utilisateur grâce au fait que WINDOWS fait la gestion de l'interface pour nous. Nous nous sommes plus occupés de la réalisation de l'intégration proprement dite que des écrans de présentation à l'utilisateur. L'interface du module implémenté dans TRAMIS représente les schémas secondaire et primaire sous une forme textuelle qui correspond à la philosophie actuelle de présentation de TRAMIS/MASTER.
Le lecteur pourra trouver dans l'Annexe D un exemple d'une session d'intégration dans l'atelier TRAMIS.

CHAPITRE 7

CONCLUSION GENERALE

Dans ce travail, nous avons vu que le problème de l'intégration n'était pas trivial. Ce processus couvre une large gamme de possibilités dans les problèmes qu'un adminitrateur de bases de données chargé de l'intégration peut rencontrer. Il est donc souhaitable d'avoir un outil à sa disposition vu l'ampleur de la tâche et l'importance que jouent de nos jours les bases de données.

L'état de l'art réalisé a permis de mettre en lumière les différentes approches qui ont été adoptées par les spécialistes de l'intégration. Nous constatons une évolution depuis les années 80 à nos jours vers l'emploi d'un modèle de données sémantiquement riche et une préservation des bases de données d'origine. Aucune solution n'émerge cependant des recherches effectuées et peu d'outils appliquant ces découvertes sont disponibles: une des raisons en est l'abondance de représentations de la même réalité (due à la richesse du modèle de données adopté) et donc de l'impossibilité de trouver des mécanismes de résolution qui soient suffisamment "universels".

Devant l'ampleur du processus d'intégration, nous nous sommes restreints au problème proprement dit de l'intégration lors de l'élaboration de notre méthodologie d'intégration dans le cadre d'un atelier logiciel (TRAMIS). La première partie du processus d'intégration (i.e. la recherche des correspondances) est laissée à charge de l'administrateur de bases de données car cette phase est fortement subjective et implique une bonne connaissance du contexte, ce qui serait lourd à traiter dans un processus d'intégration.

La stratégie que nous avons adoptée est une politique d'intégration interactive d'un schéma dans un autre. En cas de conflits, nous adoptons la solution la plus générale pour englober le plus de cas possibles. Nous avons légitimé cette décision par le fait qu'une solution automatisable était impossible pour l'entièreté du processus et qu'il est préférable pour l'administrateur de bases de données de participer au processus s'il veut comprendre facilement le résultat final.

Notre méthodologie se situe dans l'école la plus courante: l'intégration physique globale qui préconise de fusionner les schémas conceptuels de bases de données pour obtenir un schéma global reprenant toute la sémantique des schémas intégrés. La méthodologie proposée est générale mais nous l'avons restreinte à un noyau minimal de fonctions uniquement basées sur l'équivalence. Une extension intéressante serait la réalisation d'une méthodologie comprenant l'ensemble des typologies de correspondance.

La solution réalisée dans TRAMIS nous a permis de mieux voir l'ampleur du problème de l'intégration. L'outil permet l'intégration d'un schéma (dit secondaire dans un schéma (dit primaire) sur base des correspondances d'équivalence entre objets (entité, association ou attribut) introduites par l'utilisateur. Cela permet non seulement d'impliquer l'utilisateur dans le processus d'intégration en lui laissant certains choix (choix des noms, de l'ordre d'intégration, . . .) dans la solution finale mais aussi de visualiser le processus tout au long de son exécution. L'approche interactive de l'intégration permet à l'utilisateur, grâce à la

richesse de l'atelier logiciel, de retravailler le schéma résultant pour le conformer à ses besoins.

Cet outil peut faire l'objet d'extensions ultérieures: on pourra notamment y inclure les autres typologies de correspondances. Nous nous sommes principalement préoccupés des problèmes propres à l'intégration en laissant de côté les aspects ergonomiques de l'interface. Il serait intéressant d'y remédier à l'avenir de même qu'on pourra s'intéresser plus particulièrement à la recherche des correspondances.

BIBLIOGRAPHIE

[ANSI 75]: ANSI/X3/SPARC study group on database management report. Tsichritzis ed. Fev 1975 réimprimé dans Information systems, vol 3, 1978.

[Batini & Lenzerini 84]: BATINI, C., AND LENZERINI, M. Nov. 1984. "A methodology for data schema integration in the entity relationship model". IEEE Trans. Softw. Eng. SE-10, 6 , 650-663.

[Batini & Al 86]: BATINI, C., LENZERINI, M., NAVATHE, S.B. Dec 1986. "A comparative analysis of methodologies for database schema integration". ACM computing survey, Vol 18, N°4.

[Belfar 84]: BELFAR, K. 1984. "Méthode d'intégration de schémas et application aux schémas exprimés dans un modèle de type E-R-A". Thèse de doctorat, Université Pierre et Marie Curie (PARIS VI).

[Biskup & Convent 86]: BISKUP, J., CONVENT, B. 1987. "A formal view integration method". ACM-SIGMOD.

[Bodart & Pigneur 89]: BODART, F., PIGNEUR, Y. 1989. "Conception assistée des systèmes d'information. Méthode - Modèle - Outils". 2° édition. Edition Masson.

[Bouzeghoub 84]: BOUZEGHOUB, M. 1984. "MORSE: A functional query language and its semantic data model". Proc of 84 Trends and Application conf. on Databases, IEEE-NBS Gaithersburg (USA).

[Bouzeghoub & Al 90]: BOUZEGHOUB, M., COMYN-WATTIAU, I. October 8-10 1990. "View integration by semantic unification and transformation of data structures", 9th international conference on entity-relationship, Lausanne(ch).

[Brès 90]: BRÈS, P.-A. Sept. 1990. "Tramis: concevoir une base de données, c'est la base d'un système d'information". Génie logiciel & Système expert, N° 20, p62-66.

[Casanova & Vidal 83]: CASANOVA, M., AND VIDAL, M. 1983. "Towards a sound view integration methodology". In Proceedings of the 2nd ACM SIGACT/SIGMOD Conference on Principles of Database Systems (Altanta, Ga., Mar. 21-23). ACM New York, pp. 36-47.

[Celis & Deudon 88]: CELIS, M., Deudon, E. 1988. "Intégration de schémas conceptuels ERA", mémoire de fin d'études (2 vol), Facultés universiaires Notre-Dame de la Paix, Namur.

[Chen 76]: CHEN, P.P. mars 1976. "The entity relationship model - Toward a unified view of data", ACM TODS, Vol. 1, N°1.

[Codd 71]: CODD, E.,F. November 1971. "A database sublanguage based on the relational calculus", in proc. ACM-SIGFODET Workshop Data Description, Access and Control, San Diego, CA, pp 35-38.

[Comyn-Wattiau 90]: COMYN-WATTIAU, I. 22 JUIN 1990. "L'intégration de vues dans le système expert SECSI". Thèse de doctorat à l'université Paris VI.

[Dayal & Hwang 84]: DAYAL, U., HWANG, H.Y. Nov 1984. "View definition and generalization for database integration in a multidatabase system". IEEE Transactions On Software Engineering, SE Vol. 10, N°6.

[Elmasri & Wiederhold 79]: ELMASRI, R., WIEDERHOLD, G. 1979. "Data model integration using the structural model". ACM SIGMOD.

[Frantz 90]: FRANTZ, G. July 1990."Microsoft Windows 3.0 Guide du programmeur". Sybex, système d'exploitation.

[Hainaut 86a]: HAINAUT, J.-L. 1986. "Conception assistée des applications informatiques, partie No 2 conception de la base de données". Edition Massons Presses Universitaires de Namur 1986.

[Hainaut 86b]: HAINAUT, J.-L. 20 Août 1986. "Schémas de la base des spécifications". Projet Atelier Bases de Données. Spécification - SPEC-86/8-4.

[Hainaut 88]: HAINAUT, J.-L. 1988. "Introduction à la théorie relationnelle des bases de données". Notes de cours provisoires.

[Hainaut 89]: HAINAUT, J.-L. 1989. "Bases de données et bases de connaissances en gestion des organisations". Notes de cours provisoires.

[Hainaut 90]: HAINAUT, J.-L. 1990. "TRAMIS: a transformation-based database CASE tool". Paper written at the Faculty N-D de la Paix, Namur.

[Jardine 89]: JARDINE, D.A., YAZID, S. 1989. "Integration of Information Submodels". Information Systems Concepts: An In-Depth Analysis, E.D. Falkenberg and P. Lindgreen Eds., North-Holland.

[Khan 79]: KHAN, B. 1979. "A structured logical database design methodology". Ph.D. dissertation, Computer Science Dept., Univ. of Michigan, Ann Arbor, Mich.

[Larson & Al 89]: LARSON, J.A., NAVATHE, S.B., EL-MASRI, R. Avril 1989. "A theory of attribute equivalence in databases with application to schema integration", IEEE Transactions On Software Engineering, Vol. 15, N°4.

[Litwin 84]: LITWIN, W. Apr. 24-27 1984. "MALPHA: A relational multidatabase manipulation language". in proc. IEEE Comput. Soc. First Int. Conf. Data Eng., Los Angeles, CA, pp 86-93.

[Motro 87]: MOTRO, A. Juli 1987. "Superviews: virtual integration of multiple databases", IEEE Transactions On Software Engineering, Vol. 13, N°7.

[Navathe & Gadgil 82]: NAVATHE, S.B., AND GADGIL, S.G. 1982. "A methodology for view integration in logical database design". In Proceedings of the 8th International Conference on Very Large DataBases (Mexico City). VLDB Endowment, Saratoga, Calif.

[Spaccapietra & Parent 90]: SPACCAPIETRA, S., PARENT, C. 1990. "View integration: a step forward", Ecole Polytechnique de Lausanne, laboratoire Bases de données - IN - ECUBLENS.

[Tardieu & Al 79]: TARDIEU, H., NANCI, D., PASCOT, D. 1979. "Conception de système d'information: Conception de la base de données". Editions d'Organisation, Paris.

[Teory & Fry 82]: TEOREY, T., AND FRY, J. 1982. "Design of database structures". Prentice-Hall, Englewood Cliffs, N.J.

[Yao & Al 82]: YAO, S. B., WADDLE, V., AND HOUSEL, B. 1982. "View modeling and integration using the functional data model". IEEE Trans. Softw. Eng. SE-8, 6, 544-553.

FACULTES UNIVERSITAIRES N.D. DE LA PAIX NAMUR

INSTITUT D'INFORMATIQUE

INTEGRATION DE SPECIFICATIONS DE BASES DE DONNEES

ANNEXES

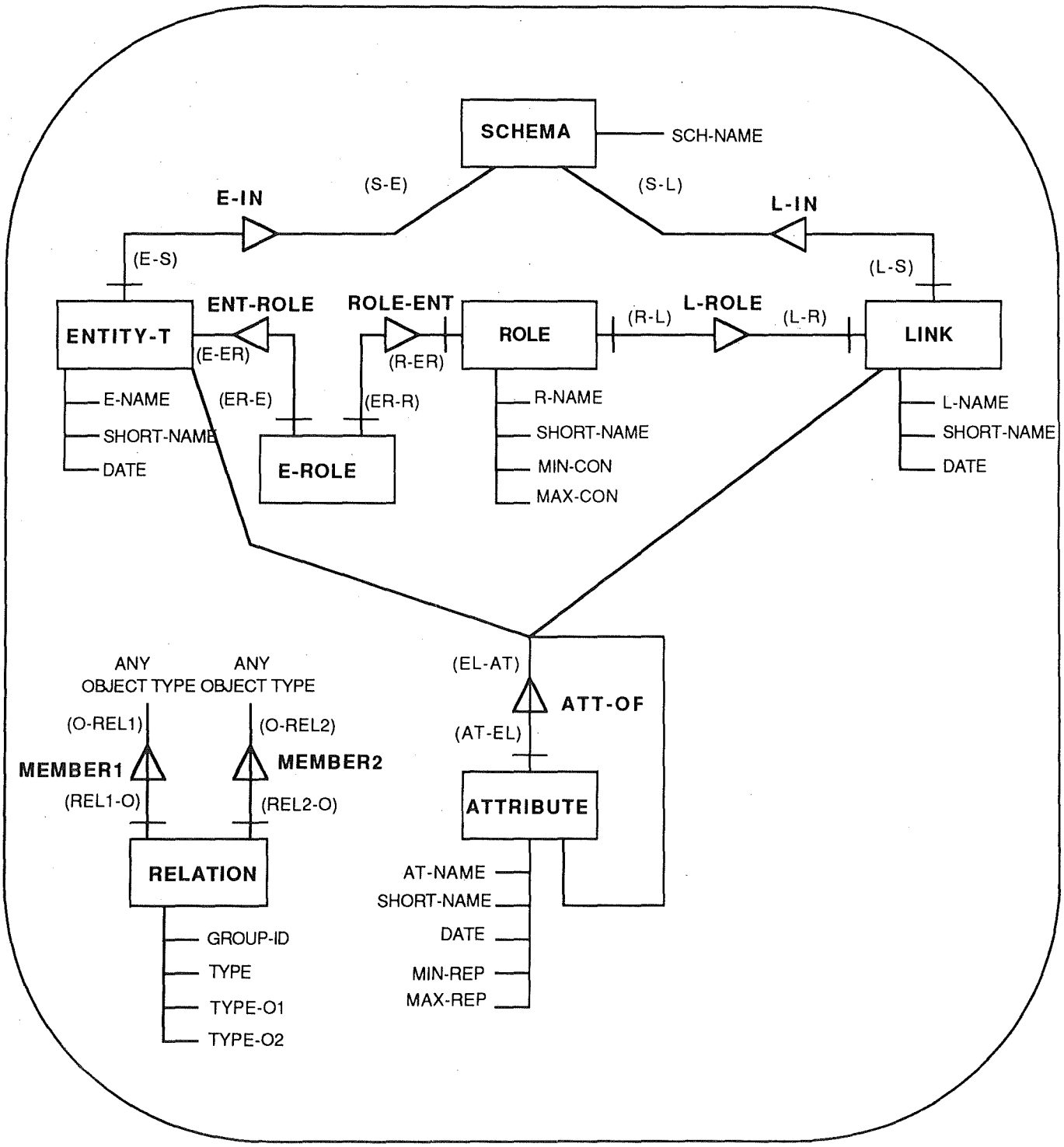
Vincent MYNSBERGHE
Patrick VANACKERE

Mémoire présenté sous la direction du
Professeur Jean-Luc HAINAUT
Pour l'obtention du titre de
Licencié et Maître en Informatique

Année académique 1990-1991

ANNEXE A

MÉTA-SCHÉMA DE L'ATELIER TRAMIS/MASTER



ANNEXE B

MÉTA-ALGORITHME DU MODULE D'INTÉGRATION

B.1 Fonction Principale du module d'intégration

```
procedure integration;
{
Spécification:

Variables globales du module d'intégration:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'attribut de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'association de Sp à intégrer avec
                   ObjectInt1.
     TypeObjectInt1 : Type de ObjectInt1 ('A', 'R' ou 'E');
     TypeObjectInt1 : Type de ObjectInt2 ('A', 'R' ou 'E');
     PereObjectInt1 : nom du père de ObjectInt1,
     Niveau1     : niveau de ObjectInt1 dans la liste des attributs
                   de PereObjectInt1.
     PereObjectInt2 : nom du père de ObjectInt2,
     Niveau2     : niveau de ObjectInt2 dans la liste des attributs
                   de PereObjectInt2.

But: Cette Procédure réalise l'intégration d'un schéma
     (dit secondaire Ss) dans un autre schéma (dit primaire Sp).

Précondition:
                /
Postcondition:

   L'outil a intégré toutes les correspondances introduites par
   l'utilisateur.
}
begin
    if (count(SCHEMA) < 2)
    then
        display "Pour réaliser l'intégration, il faut
                 au moins deux schémas !";
        return;
    endif;

    CurSchName1:= '';
    CurSchName2:= '';
    ObjectInt1:= '';
    ObjectInt2:= '';
    TypeObjectInt1:= '';
    TypeObjectInt2:= '';
    PereObjectInt1:= '';
    PereObjectInt2:= '';
    Niveau1:= '';
    Niveau2:= '';

    select(CurSchName1);
    select(CurSchName2);
```

```
while PAS_FIN_DE_L'INTEGRATION do
   select(ObjectInt1,TypeObjectInt1,PereObjectInt1,Niveau1);
   select(ObjectInt1,TypeObjectInt1,PereObjectInt1,Niveau1);

   case TypeObjectInt1 of
      'E': case TypeObjectInt2 of
              'E': if (EE_precond(mess_err)== TRUE)
                   then
                       EE_integ;
                   else
                       EE_aff_err(mess_err);
                   endif;
              break;
              'R': if (ER_precond(mess_err)== TRUE)
                   then
                       ER_integ;
                   else
                       ER_aff_err(mess_err);
                   endif;
              break;
              'A': if (EA_precond(mess_err)== TRUE)
                   then
                       EA_integ;
                   else
                       EA_aff_err(mess_err);
                   endif;
              break;
              'N': if (EN_precond(mess_err)== TRUE)
                   then
                       EN_integ;
                   else
                       EN_aff_err(mess_err);
                   endif;
              break;
         endcase;
      break;

      'R': case TypeObjectInt2 of
              'E': if (RE_precond(mess_err)== TRUE)
                   then
                       RE_integ;
                   else
                       RE_aff_err(mess_err);
                   endif;
              break;
              'R': if (RR_precond(mess_err)== TRUE)
                   then
                       RR_integ;
                   else
                       RR_aff_err(mess_err);
                   endif;
              break;
              'A': if (RA_precond(mess_err)== TRUE)
                   then
                       RA_integ;
                   else
                       RA_aff_err(mess_err);
                   endif;
              break;
              'N': if (RN_precond(mess_err)== TRUE)
                   then
                       RN_integ;
                   else
                       RN_aff_err(mess_err);
                   endif;
```

B–2 Méta-algorithme du module d'intégration

```
                    break;
                endcase;
                break;
                'A': case TypeObjectInt2 of
                        'E': if (AE_precond(mess_err)== TRUE)
                             then
                                 AE_integ;
                             else
                                 AE_aff_err(mess_err);
                             endif;
                        break;
                        'R': if (AR_precond(mess_err)== TRUE)
                             then
                                 AR_integ;
                             else
                                 AR_aff_err(mess_err);
                             endif;
                        break;
                        'A': if (AA_precond(mess_err)== TRUE)
                             then
                                 AA_integ;
                             else
                                 AA_aff_err(mess_err);
                             endif;
                        break;
                        'N': if (AN_precond(mess_err)== TRUE)
                             then
                                 AN_integ;
                             else
                                 AN_aff_err(mess_err);
                             endif;
                        break;
                   endcase;
                break;
            endcase;

            ObjectInt1:= '';
            ObjectInt2:= '';
            TypeObjectInt1:= '';
            TypeObjectInt2:= '';
            PereObjectInt1:= '';
            PereObjectInt2:= '';
            Niveau1:= '';
            Niveau2:= '';
        endwhile;
    end;
```

B.2 Fonctions EE

Fonction: EE_aff_err

```
procedure EE_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance EE.

Précondition:

     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'E',
     mess_err : numéro du message d'erreur (compris entre 1 et 2).
```

```
Postcondition:

     Si mess_err = 1, le message "L'entité sélectionnée dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "L'entité sélectionnée dans
                      le schéma primaire a déjà été mise en
                      correspondance" est présenté à l'utilisateur;
}
begin
 if mess_err = 1
 then
   display "L'entité sélectionnée dans le schéma secondaire a déjà
            un correspondant";
 else
   display "L'entité sélectionnée dans le schéma primaire a déjà été
            mise en correspondance";
 endif;
end;
```

Fonction: EE_Integ

```
procedure EE_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section 5.5.2 pour l'intégration de
     l'entité sélectionnée dans le schéma secondaire avec l'entité
     sélectionnée dans le schéma primaire.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'entité de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'entité de Sp à intégrer avec ObjectInt1,
     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'E'.

Postcondition:

     MS(Ss,Sp)= MS + EE ObjectInt1*Equivalent*ObjectInt2.
}
begin
   Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
   Ent1:= ENTITY_T((E_S: Sch1)
                   and (: E_NAME = ObjectInt1));
   Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
   Ent2:= ENTITY_T((E_S: Sch2)
                   and (: E_NAME = ObjectInt2));

   create rel:= RELATION ((REL1_O: Ent1)
                          and (REL2_O: Ent2)
                          and (: GROUP_ID = count(RELATION) + 1)
                          and (: TYPE = "INT")
                          and (: TYPE_O1 = "E")
                          and (: TYPE_O2 = "E"));

   if (E_NAME(: Ent1) <> E_NAME(: Ent2))
   then
      if (libelle_unique(E_NAME(: Ent1)) = TRUE)
      then
          accept "Voulez-vous intervertir les ",resultat;
          if (resultat = OUI)
          then
              modify Ent2(:E_NAME = E_NAME(: Ent1));
          endif;
end;
```

Fonction: EE_precond

```
function EE_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'intégration d'une entité de Ss avec une entité de Sp
     sont remplies. Ces conditions ont fait l'objet d'une
     description à la section 5.5.2.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'entité de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'entité de Sp à intégrer avec ObjectInt1,
     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'E'.

Postcondition:

     EE_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
                        mess_err = 2 si ObjectInt2 a déjà été mis
                                     en correspondance avec un objet
                                     de Ss.
}
begin
  Sch:= SCHEMA(: SCH_NAME = CurSchName1);
  Ent:= ENTITY_T((E_S: Sch)
                 and (: E_NAME = ObjectInt1));
  if (RELATION((REL1_O: Ent)
               and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;

  Sch:= SCHEMA(: SCH_NAME = CurSchName2);
  Ent:= ENTITY_T((E_S: Sch)
                 and (: E_NAME = ObjectInt2));
  if (RELATION((REL2_O: Ent)
               and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 2;
      return(FALSE);
  endif;
  return(TRUE);
end;
```

B.3 Fonctions ER

Fonction: ER_aff_err

```
procedure ER_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance ER.

Précondition:

     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'R',
     mess_err : numéro du message d'erreur (compris entre 1 et 2).

Postcondition:
```

```
     Si mess_err = 1, le message "L'entité sélectionnée dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "L'association sélectionnée dans
                      le schéma primaire a déjà été mise en
                      correspondance" est présenté à l'utilisateur;
)
begin
 if mess_err = 1
 then
   display "L'entité sélectionnée dans le schéma secondaire a déjà
            un correspondant";
 else
   display "L'association sélectionnée dans le schéma primaire a déjà
            été mise en correspondance";
 endif;
end;
```

Fonction: ER_Integ

```
procedure ER_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section 5.5.2 pour l'intégration de
     l'entité sélectionnée dans le schéma secondaire avec
     l'association sélectionnée dans le schéma primaire.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'entité de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'association de Sp à intégrer avec
                   ObjectInt1,
     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'R'.

Postcondition:

     MS(Ss,Sp)= MS + EE ObjectInt1*Equivalent*ObjectInt2.
}
begin
   Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
   Ent:= ENTITY_T((E_S: Sch1)
                  and (:E_NAME = ObjectInt1));
   Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
   Lk:= LINK((L_S: Sch2)
             and (:L_NAME = ObjectInt2));

   create EntLk:= ENTITY_T((: E_NAME = L_NAME(: Lk))
                           and (: SHORT_NAME = SHORT_NAME(: Lk)
                           and (: DATE = DATE(: Lk)
                           and (E_S: Sch2);

   Att:= first(ATTRIBUTE(AT_EL: Lk));
   while Att <> () do
     modify Att(At_EL: EntLk);
     Att:= next(ATTRIBUTE(AT_EL: Lk));
   endwhile;

   Rol:= first(ROLE(R_L: Lk));
   while Rol <> () do
     create Lk1:= LINK((L_S: Sch2)
                       and (: L_NAME = choisi par le système
                                       d'intégration)
                       and (: SHORT_NAME = les trois premiers
                                           caractères de L_NAME)
                       and (: DATE = date du système));
```

```
    create Rol1:= ROLE((: R_NAME = choisi par le système
                                   d'intégration)
                       and (: SHORT_NAME = les trois premiers
                                           caractères de R_NAME)
                       and (: DATE= date du système)
                       and (: MIN_CON = 1)
                       and (: MAX_CON = 1)
                       and (R_L : Lk1));
    create Erol:= E_ROLE((ER_E: EntLk)
                         and (ER_R: Rol1));
    Rol:= next(ROLE(R_L: Lk));
  endwhile;
  Rel:= RELATION((REL2_O: Lk)
                 (: TYPE = "INT"));

  if (Rel <> ())
  then
     modify Rel(REL2_O: EntLk);
  endif;

  delete Lk;

  EE_integ; /* Intégration de Ent avec EntLk */
end;
```

Fonction: ER_precond

```
function ER_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'intégration d'une entité de Ss avec une association de Sp
     sont remplies. Ces conditions ont fait l'objet d'une
     description à la section 5.5.2.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'entité de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'association de Sp à intégrer avec
                   ObjectInt1,
     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'R'.

Postcondition:

     ER_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
                        mess_err = 2 si ObjectInt2 a déjà été mis
                                     en correspondance avec un objet
                                     de Ss.
}
begin
  Sch:= SCHEMA(: SCH_NAME = CurSchName1);
  Ent:= ENTITY_T((E_S: Sch)
                 and (: E_NAME = ObjectInt1));
  if (RELATION((REL1_O: Ent)
               and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;

  Sch:= SCHEMA(:SCH_NAME = CurSchName2);
  Lk := LINK((L_S: Sch)
             and (: L_NAME = ObjectInt2));
  if (RELATION((REL2_O: Ent)
               and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 2;
      return(FALSE);
  endif;
  return(TRUE);
end;
```

B.4 Fonctions EA

Fonction: EA_aff_err

```
procedure EA_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance EA.

Précondition:

     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'A',
     mess_err : numéro du message d'erreur (compris entre 1 et 3).

Postcondition:

     Si mess_err = 1, le message "L'entité sélectionnée dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "L'attribut sélectionné dans
                      le schéma primaire a déjà été mis en
                      correspondance" est présenté à l'utilisateur;
        mess_err = 3, le message "L'attribut sélectionné dans le
                      schéma primaire n'est pas de niveau 1"
}
begin
 if mess_err = 1
 then
   display "L'entité sélectionnée dans le schéma secondaire a déjà
            un correspondant";
 else
   if mess_err = 2
      display "L'attribut sélectionné dans le schéma primaire a déjà
               été mis en correspondance";
   else
      display "L'attribut sélectionné dans le schéma primaire n'est
               pas de niveau 1";
   endif;
 endif;
end;
```

Fonction: EA_integ

```
procedure EA_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section 5.5.2 pour l'intégration de
     l'entité sélectionnée dans le schéma secondaire avec
     l'attribut sélectionné dans le schéma primaire.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'entité de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'attribut de Sp à intégrer avec
                   ObjectInt1,
     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'A',
     PereObjectInt2 : nom du père de ObjectInt2,
     Niveau2     : niveau de ObjectInt2 dans la liste des attributs
                   de PereObjectInt2.

Postcondition:
```

```
    MS(Ss,Sp)= MS + EE ObjectInt1*Equivalent*ObjectInt2.
}
begin
   Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
   Ent:= ENTITY_T((E_S: Sch1)
                  and (:E_NAME = ObjectInt1));
   Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
   Type := Rch_type_pere(PereObjectInt2, Sch);
   if (Type = "E")
   then
     Pere:= ENTITY_T(: E_NAME = PereObjectInt2);
     Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
   else
     Pere:= LINK(: L_NAME = PereObjectInt2);
     Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
   endif;

   if (Type = "R")
   then
       create EntLk:= ENTITY_T((: E_NAME = L_NAME(: Pere))
                               and (: SHORT_NAME = SHORT_NAME(: Pere)
                               and (: DATE = DATE(: Pere)
                               and (E_S: Sch2);

       Att:= first(ATTRIBUTE(AT_EL: Pere));
       while Att <> () do
          modify Att(At_EL: EntLk);
          Att:= next(ATTRIBUTE(AT_EL: Pere));
       endwhile;

       Rol:= first(ROLE(R_L: Pere));
       while Rol <> () do
        create Lk1:= LINK((L_S: Sch2)
                          and (: L_NAME = choisi par le système
                                          d'intégration)
                          and (: SHORT_NAME = les trois premiers
                                              caractères de L_NAME)
                          and (: DATE = date du système));
        create Rol1:= ROLE((: R_NAME = choisi par le système
                                       d'intégration)
                           and (: SHORT_NAME = les trois premiers
                                               caractères de R_NAME)
                           and (: DATE= date du système)
                           and (: MIN_CON = 1)
                           and (: MAX_CON = 1)
                           and (R_L: Lk1));
        create Erol:= E_ROLE((ER_E: EntLk)
                             and (ER_R: Rol1));
        Rol:= next(ROLE(R_L: Pere));
      endwhile;

      Rel:= RELATION((REL2_O: Pere)
                     (: TYPE = "INT"));
      if (Rel <> ())
      then
         modify Rel(REL2_O: EntLk);
      endif;

      delete Pere;

      Ent2:= EntLK;
   else
      Ent2:= Pere;
   endif;

   Create EntAtt:= ENTITY_T((: E_NAME = AT_NAME(: Att))
                            and (: SHORT_NAME = SHORT_NAME(: Att))
                            and (: DATE = DATE(: Att))
                            and (E_S: Sch2));

   Create Att1:= ATTRIBUTE((: AT_NAME = At_NAME(: Att))
                           and (: SHORT_NAME = SHORT_NAME(: Att))
                           and (: DATE = DATE(: Att))
                           and (: MIN_REP = 1);
                           and (: MAX_REP = 1);
                           and (AT_EL: EntAtt));

   Create Lk:= LINK ((: L_NAME = choisi par le système
                                 d'intégration)
                     and (: SHORT_NAME = les trois premiers
                                         caractères de L_NAME)
                     and (: DATE = date du système)
                     and (L_S: Sch2));
```

```
   Create Rol1:= ROLE ((: R_NAME = choisi par le système
                                   d'intégration)
                       and (: SHORT_NAME = les trois premiers
                                           caractères de R_NAME)
                       and (: DATE = date du système)
                       and (: MIN_CON = MIN_REP(: Att))
                       and (: MAX_CON = MAX_REP(: Att))
                       and (R_L: Lk));

   Create Erol1:= E_ROLE ((ER_R: Rol1)
                          and (ER_E: Ent2));

   Create Rol2:= ROLE ((: R_NAME = choisi par le système
                                   d'intégration)
                       and (: SHORT_NAME = les trois premiers
                                           caractères de R_NAME)
                       and (: DATE = date du système)
                       and (: MIN_CON = 1)
                       and (: MAX_CON = MAX_REP(: Att))
                       and (R_L: Lk));

   Create Erol2:= E_ROLE ((ER_R: Rol2)
                          and (ER_E: EntAtt));
   delete Att;

   EE_integ; /* Intégration de Ent avec EntAtt */
end;
```

Fonction: EA_precond

```
function EA_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'intégration d'une entité de Ss avec un attribut de Sp
     sont remplies. Ces conditions ont fait l'objet d'une
     description à la section 5.5.2.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'entité de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'attribut de Sp à intégrer avec ObjectInt1,
     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'A',
     PereObjectInt2 : nom du père de ObjectInt2,
     Niveau2     : niveau de ObjectInt2 dans la liste des attributs
                   de PereObjectInt2.

Postcondition:

     EA_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
                        mess_err = 2 si ObjectInt2 a déjà été mis
                                     en correspondance avec un objet
                                     de Ss;
                        mess_err = 3 si ObjectInt2 n'est pas un
                                     attribut de niveau 1.
}
begin
  Sch:= SCHEMA(: SCH_NAME = CurSchName1);
  Ent:= ENTITY_T((E_S: Sch)
                 and (: E_NAME = ObjectInt1));
  if (RELATION((REL1_O: Ent)
               and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;
```

```
  Sch:= SCHEMA(: SCH_NAME = CurSchName2);
  Type := Rch_type_pere(PereObjectInt2, Sch);
  if (Type = "E")
  then
    Pere:= ENTITY_T(: E_NAME = PereObjectInt2);
    Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
  else
    Pere:= LINK(: L_NAME = PereObjectInt2);
    Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
  endif;

  if (RELATION((REL2_O: Att)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 2;
      return(FALSE);
  endif;

  if (niveau2 <> 1)
  then
      mess_err:=3;
      return(FALSE);
  endif;
  return(TRUE);
end;
```

B.5 Fonctions EN

Fonction: EN_aff_err

```
procedure EN_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance EN.

Précondition:

     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'N',
     mess_err : numéro du message d'erreur (1 uniquement).

Postcondition:

     Si mess_err = 1, le message "L'entité sélectionnée dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur.
}
begin
 if mess_err = 1
 then
   display "L'entité sélectionnée dans le schéma secondaire a déjà
            un correspondant";
 endif;
end;
```

Fonction: EN_Integ

```
procedure EN_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section 5.5.2 pour l'ajout de
     l'entité sélectionnée dans le schéma secondaire dans le schéma
     primaire.

Précondition:
```

```
     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'entité de Ss à ajouter dans Sp,
     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'N'.

Postcondition:

     MS(Ss,Sp)= MS + EE ObjectInt1*Equivalent*ObjectInt1.
}
begin
   Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
   Ent1:= ENTITY_T((E_S: Sch1)
                   and (: E_NAME = ObjectInt1));
   Sch1:= SCHEMA(: SCH_NAME = CurSchName2);

   ObjectInt2:= ObjectInt1;
   create Ent2:= ENTITY_T((: E_NAME = E_NAME(: Ent1))
                          and (: SHORT_NAME = SHORT_NAME(: Ent1))
                          and (: DATE = DATE(: Ent1))
                          and (E_S: Sch2));
   EE_integ; /* Intégration de Ent1 avec Ent2 */
end;
```

Fonction: EN_precond

```
function EN_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'ajout d'une entité de Ss dans Sp sont remplies. Ces
     conditions ont fait l'objet d'une description à la
     section 5.5.2.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     ObjectInt1  : nom de l'entité de Ss à intégrer dans Sp,
     TypeObjectInt1 = 'E',
     TypeObjectInt2 = 'N'.

Postcondition:

     EN_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
}
begin
  Sch:= SCHEMA(: SCH_NAME = CurSchName1);
  Ent:= ENTITY_T((E_S: Sch)
                 and (: E_NAME = ObjectInt1));
  if (RELATION((REL1_O: Ent)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;
  return(TRUE);
```

B.6 Fonctions RR

Fonction: RR_aff_err

```
procedure RR_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance RR.

Précondition:

     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'R',
     mess_err : numéro du message d'erreur (compris entre 1 et 4).

Postcondition:

     Si mess_err = 1, le message "L'association sélectionnée dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "L'association sélectionnée dans
                      le schéma primaire a déjà été mise en
                      correspondance" est présenté à l'utilisateur;
        mess_err = 3, le message "Une des entités participant à
                      l'association du schéma secondaire n'a pas
                      encore été mise en correspondance" est
                      présenté à l'utilisateur;
        mess_err = 4, le message "Un des correspondants des entités
                      dans le schéma primaire n'a pas un chemin le
                      reliant à l'association" est présenté
                      à l'utilisateur.
}
begin
 if mess_err = 1
 then
   display "L'association sélectionnée dans le schéma secondaire a
            déjà un correspondant";
 else
   if mess_err = 2
   then
        display "L'association sélectionnée dans le schéma primaire a
                 déjà été mise en correspondance";
   else
     if mess_err = 3
     then
        display "Une des entités participant à l'association du schéma
                 secondaire n'a pas encore été mise en
                 correspondance";
     else
        display "Un des correspondants des entités dans schéma
                 primaire n'a pas un chemin le reliant à
                 l'association";
     endif;
   endif;
 endif;
end;
```

Fonction: RR_Integ

```
procedure RR_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section 5.5.3 pour l'intégration de
     l'association sélectionnée dans le schéma secondaire avec
     l'association sélectionnée dans le schéma primaire.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'association de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'association de Sp à intégrer avec
                   ObjectInt1,
     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'R'.
```

```
Postcondition:

     MS(Ss,Sp)= MS + RR ObjectInt1*Equivalent*ObjectInt2.
}
begin
   Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
   Lk1:= LINK((L_S: Sch1)
             and (: L_NAME = ObjectInt1));
   Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
   Lk2:= LINK((L_S: Sch2)
             and (: L_NAME = ObjectInt2));

   Rol1:= first(ROLE(R_L: Lk1);
   while Rol1 <> () do
     Ent1:= ENTITY_T(E_ER: E_ROLE(ER_R: Rol1);
     Ent2:= ENTITY_T(O_REL2: RELATION ((REL1_O: Ent1)
                                       and (: TYPE = "INT")));
     Rol2:= ROLE(R_ER: E_ROLE(ER_E: Ent2));

     if (MIN_CON(: Rol1) < MIN_CON(: Rol2))
     then
       modify Rol2(: MIN_CON = MIN_CON(:ROL1));
     endif;
     if (MAX_CON(: Rol1) > MAX_CON(: Rol2))
     then
       modify Rol2(: MAX_CON = MAX_CON(:ROL1));
     endif;
     Rol1:= next(ROLE(R_L: Lk1);
   endwhile;

   create rel:= RELATION ((REL1_O: Lk1)
                          and (REL2_O: Lk2)
                          and (: GROUP_ID = count(RELATION) + 1)
                          and (: TYPE = "INT")
                          and (: TYPE_O1 = "R")
                          and (: TYPE_O2 = "R"));

   if (L_NAME(: Lk1) <> L_NAME(: Lk2))
   then
      if (libelle_unique(L_NAME(: Lk1) = TRUE)
      then
          accepte "Voulez-vous intervertir les ",resultat;
          if (resultat = OUI)
          then
              modify Lk2(:L_NAME = L_NAME(: Lk1));
          endif;
      endif;
   endif;
end;
```

Fonction: RR_precond

```
function RR_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'intégration d'une association de Ss avec une association
     de Sp sont remplies. Ces conditions ont fait l'objet d'une
     description à la section 5.5.3.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'association de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'association de Sp à intégrer avec
                   ObjectInt1,
     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'R'.

Postcondition:
```

```
    RR_precond = TRUE  si toutes les conditions sont remplies;
                       mess_err = 0.
                 FALSE si au moins une des conditions n'est pas
                       remplie;
                       mess_err = 1 si ObjectInt1 a déjà un
                                    correspondant dans Sp;
                       mess_err = 2 si ObjectInt2 a déjà été mis
                                    en correspondance avec un objet
                                    de Ss;
                       mess_err = 3 si au moins une entité
                                    participant à ObjectInt1 n'a pas
                                    encore été mis en correspondance
                                    avec un objet de Sp;
                       mess_err = 4 si au moins un des correspondants
                                    des entités participant à
                                    ObjectInt1 ne participe pas à
                                    ObjectInt2.
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Lk1:= LINK((L_S: Sch1)
             and (: L_NAME = ObjectInt1));
  if (RELATION((REL1_O: Lk1)
               and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;

  Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
  Lk2:= LINK((L_S: Sch2)
             and (: L_NAME = ObjectInt2));
  if (RELATION((REL2_O: Lk2)
               and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 2;
      return(FALSE);
  endif;

  for Ent := ENTITY_T(E_ER: E_ROLE(ER_R: ROLE(R_L: Lk1))) do
    if (RELATION((REL1_O: Ent)
                 and (: TYPE = "INT"))) = 0
    then
        mess_err:=3;
        return(FALSE);
    endif;

    Ent2:= ENTITY_T((O_REL2: RELATION((REL1_O: Ent)
                                      and (:TYPE = "INT")))
                    and (E_S: Sch2));

    if (chemin_RE(Ent2,LK2) = FALSE)
    then
        mess_err:= 4;
        return(FALSE);
    endif;
  endfor;
  return(TRUE);
end;
```

B.7 Fonctions RE

Fonction: RE_aff_err

```
procedure RE_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance RE.

Précondition:
```

```
     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'E',
     mess_err : numéro du message d'erreur (compris entre 1 et 4).

Postcondition:

     Si mess_err = 1, le message "L'association sélectionnée dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "L'entité sélectionnée dans
                      le schéma primaire a déjà été mise en
                      correspondance" est présenté à l'utilisateur;
        mess_err = 3, le message "Une des entités participant à
                      l'association du schéma secondaire n'a pas
                      encore été mise en correspondance" est
                      présenté à l'utilisateur;
        mess_err = 4, le message "Un des correspondants des entités
                      dans le schéma primaire n'a pas un chemin le
                      reliant à l'association" est présenté
                      à l'utilisateur.
}
begin
 if mess_err = 1
 then
   display "L'association sélectionnée dans le schéma secondaire a
            déjà un correspondant";
 else
   if mess_err = 2
   then
        display "L'entité sélectionnée dans le schéma primaire a
                 déjà été mise en correspondance";
   else
     if mess_err = 3
     then
        display "Une des entités participant à l'association du schéma
                 secondaire n'a pas encore été mise en
                 correspondance";
     else
        display "Un des correspondants des entités dans le schéma
                 primaire n'a pas un chemin le reliant à
                 l'association";
     endif;
   endif;
 endif;
end;
```

Fonction: RE_Integ

```
procedure RE_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section 5.5.3 pour l'intégration de
     l'association sélectionnée dans le schéma secondaire avec
     l'entité sélectionnée dans le schéma primaire.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'association de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'entité de Sp à intégrer avec ObjectInt1,
     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'E'.

Postcondition:

     MS(Ss,Sp)= MS + RE ObjectInt1*Equivalent*ObjectInt2.
}
begin
   Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
   Lk:= LINK((L_S: Sch1)
             and (: L_NAME = ObjectInt1));
   Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
   Ent:= ENTITY_T((E_S: Sch2)
             and (: E_NAME = ObjectInt2));
```

```
   create rel:= RELATION ((REL1_O: Lk)
                          and (REL2_O: Ent)
                          and (: GROUP_ID = count(RELATION) + 1)
                          and (: TYPE = "INT")
                          and (: TYPE_O1 = "R")
                          and (: TYPE_O2 = "E"));

   if (L_NAME(: Lk) <> E_NAME(: Ent))
   then
      if (libelle_unique(L_NAME(: Lk1) = TRUE)
      then
          accepte "Voulez-vous intervertir les ",resultat;
          if (resultat = OUI)
          then
              modify Ent(:E_NAME = L_NAME(: Lk1));
          endif;
      endif;
   endif;
end;

end;
```

Fonction: RE_precond

```
function RE_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'intégration d'une association de Ss avec une entité
     de Sp sont remplies. Ces conditions ont fait l'objet d'une
     description à la section 5.5.3.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'association de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'entité de Sp à intégrer avec ObjectInt1,
     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'E'.

Postcondition:

     RE_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
                        mess_err = 2 si ObjectInt2 a déjà été mis
                                     en correspondance avec un objet
                                     de Ss;
                        mess_err = 3 si au moins une entité
                                     participant à ObjectInt1 n'a pas
                                     encore été mis en correspondance
                                     avec un objet de Sp;
                        mess_err = 4 si au moins un des correspondants
                                     des entités participant à
                                     ObjectInt1 n'est pas relié à
                                     ObjectInt2 via une association.
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Lk:= LINK((L_S: Sch1)
            and (: L_NAME = ObjectInt1));
  if (RELATION((REL1_O: Lk)
               and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;
```

```
  Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
  Ent:= ENTITY_T((E_S: Sch2)
                  and (: E_NAME = ObjectInt2));
  if (RELATION((REL2_O: Ent)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 2;
      return(FALSE);
  endif;

  for Ent1 := ENTITY_T(E_ER: E_ROLE(ER_R: ROLE(R_L: Lk))) do
    if (RELATION((REL1_O: Ent1)
                  and (: TYPE = "INT"))) = 0
    then
        mess_err:=3;
        return(FALSE);
    endif;

    Ent2:= ENTITY_T((O_REL2: RELATION((REL1_O: Ent1)
                                      and (:TYPE = "INT")))
                    and (E_S: Sch2));

    if (chemin_EE(Ent,Ent2) = FALSE)
    then
        mess_err:= 4;
        return(FALSE);
    endif;
  endfor;
  return(TRUE);
end;
```

B.8 Fonctions RA

Fonction: RA_aff_err

```
procedure RA_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance RA.

Précondition:

     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'A',
     mess_err : numéro du message d'erreur (compris entre 1 et 4).

Postcondition:

     Si mess_err = 1, le message "L'association sélectionnée dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "L'attribut sélectionné dans
                      le schéma primaire a déjà été mis en
                      correspondance" est présenté à l'utilisateur;
        mess_err = 3, le message "Une des entités participant à
                      l'association du schéma secondaire n'a pas
                      encore été mise en correspondance" est
                      présenté à l'utilisateur;
        mess_err = 4, le message "Un des correspondants des entités
                      dans le schéma primaire n'a pas un chemin le
                      reliant à l'association" est présenté
                      à l'utilisateur.
}
begin
 if mess_err = 1
 then
   display "L'association sélectionnée dans le schéma secondaire a
            déjà un correspondant";
 else
   if mess_err = 2
   then
        display "L'attribut sélectionné dans le schéma primaire a
                 déjà été mis en correspondance";
```

```
   else
     if mess_err = 3
     then
        display "Une des entités participant à l'association du schéma
                 secondaire n'a pas encore été mise en
                 correspondance";
     else
        display "Un des correspondants des entités dans schéma
                 primaire n'a pas un chemin le reliant à
                 l'association";
     endif;
   endif;
 endif;
end;
```

Fonction: RA_integ

```
procedure RA_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section 5.5.2 pour l'intégration de
     l'association sélectionnée dans le schéma secondaire avec
     l'attribut sélectionné dans le schéma primaire.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'association de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'attribut de Sp à intégrer avec
                   ObjectInt1,
     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'A',
     PereObjectInt2 : nom du père de ObjectInt2,
     Niveau2     : niveau de ObjectInt2 dans la liste des attributs
                   de PereObjectInt2.

Postcondition:

     MS(Ss,Sp)= MS + RE ObjectInt1*Equivalent*ObjectInt2.
}
begin
   Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
   LK:= LINK((L_S: Sch1)
             and (:L_NAME = ObjectInt1));
   Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
   Type := Rch_type_pere(PereObjectInt2, Sch);
   if (Type = "E")
   then
     Pere:= ENTITY_T(: E_NAME = PereObjectInt2);
     Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
   else
     Pere:= LINK(: L_NAME = PereObjectInt2);
     Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
   endif;


   if (Type = "R")
   then
       create EntLk:= ENTITY_T((: E_NAME = L_NAME(: Pere))
                               and (: SHORT_NAME = SHORT_NAME(: Pere)
                               and (: DATE = DATE(: Pere)
                               and (E_S: Sch2);

       Att:= first(ATTRIBUTE(AT_EL: Pere));
       while Att <> () do
          modify Att(At_EL: EntLk);
          Att:= next(ATTRIBUTE(AT_EL: Pere));
       endwhile;

       Rol:= first(ROLE(R_L: Pere));
       while Rol <> () do

        create Lk1:= LINK((L_S: Sch2)
                          and (: L_NAME = choisi par le système
                                          d'intégration)
                          and (: SHORT_NAME = les trois premiers
                                              caractères de L_NAME)
                          and (: DATE = date du système));
```

```
      create Rol1:= ROLE((: R_NAME = choisi par le système
                                    d'intégration)
                         and (: SHORT_NAME = les trois premiers
                                             caractères de R_NAME)
                         and (: DATE= date du système)
                         and (: MIN_CON = 1)
                         and (: MAX_CON = 1)
                         and (R_L: Lk1));

      create Erol:= E_ROLE((ER_E: EntLk)
                           and (ER_R: Rol1));

      Rol:= next(ROLE(R_L: Pere));
    endwhile;

    Rel:= RELATION((REL2_O: Pere)
                   (: TYPE = "INT"));

    if (Rel <> ())
    then
       modify Rel(REL2_O: EntLk);
    endif;

    delete Pere;

    Ent2:= EntLK;
  else
    Ent2:= Pere;
  endif;

  Create EntAtt:= ENTITY_T((: E_NAME = AT_NAME(: Att))
                           and (: SHORT_NAME = SHORT_NAME(: Att))
                           and (: DATE = DATE(: Att))
                           and (E_S: Sch2));

  Create Att1:= ATTRIBUTE((: AT_NAME = At_NAME(: Att))
                          and (: SHORT_NAME = SHORT_NAME(: Att))
                          and (: DATE = DATE(: Att))
                          and (: MIN_REP = 1);
                          and (: MAX_REP = 1);
                          and (AT_EL: EntAtt));

  Create Lk:= LINK ((: L_NAME = choisi par le système
                                d'intégration)
                    and (: SHORT_NAME = les trois premiers
                                        caractères de L_NAME)
                    and (: DATE = date du système)
                    and (L_S: Sch2));

  Create Rol1:= ROLE ((: R_NAME = choisi par le système
                                  d'intégration)
                      and (: SHORT_NAME = les trois premiers
                                          caractères de R_NAME)
                      and (: DATE = date du système)
                      and (: MIN_CON = MIN_REP(: Att))
                      and (: MAX_CON = MAX_REP(: Att))
                      and (R_L: Lk));

  Create Erol1:= E_ROLE ((ER_R: Rol1)
                         and (ER_E: Ent2));


  Create Rol2:= ROLE ((: R_NAME = choisi par le système
                                  d'intégration)
                      and (: SHORT_NAME = les trois premiers
                                          caractères de R_NAME)
                      and (: DATE = date du système)
                      and (: MIN_CON = 1)
                      and (: MAX_CON = MAX_REP(: Att))
                      and (R_L: Lk));

  Create Erol2:= E_ROLE ((ER_R: Rol2)
                         and (ER_E: EntAtt));

  delete Att;

  ER_integ; /* Intégration de Lk avec EntAtt */
end;
```

Fonction: RA_precond

```
function RA_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'intégration d'une association de Ss avec un attribut
     de Sp sont remplies. Ces conditions ont fait l'objet d'une
     description à la section 5.5.3.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'association de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'attribut de Sp à intégrer avec ObjectInt1.
     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'A',
     PereObjectInt2 : nom du père de ObjectInt2,
     Niveau2      : niveau de ObjectInt2 dans la liste des attributs
                    de PereObjectInt2.

Postcondition:

     RA_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
                        mess_err = 2 si ObjectInt2 a déjà été mis
                                     en correspondance avec un objet
                                     de Ss;
                        mess_err = 3 si au moins une entité
                                     participant à ObjectInt1 n'a pas
                                     encore été mis en correspondance
                                     avec un objet de Sp;
                        mess_err = 4 si au moins un des correspondants
                                     des entités participant à
                                     ObjectInt1 n'est pas relié à
                                     ObjectInt2 via une association.
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Lk:= LINK((L_S: Sch1)
            and (: L_NAME = ObjectInt1));
  if (RELATION((REL1_O: Lk)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;

  Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
  Type := Rch_type_pere(PereObjectInt2, Sch2);
  if (Type = "E")
  then
    Pere:= ENTITY_T(: E_NAME = PereObjectInt2);
    Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
  else
    Pere:= LINK(: L_NAME = PereObjectInt2);
    Rch_attribute(ObjectInt2,Niveau2,Pere,Att);
  endif;

  if (RELATION((REL2_O: Att)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 2;
      return(FALSE);
  endif;

  for Ent2 := ENTITY_T(E_ER: E_ROLE(ER_R: ROLE(R_L: Lk))) do
    if (RELATION((REL1_O: Ent2)
                and (: TYPE = "INT"))) = 0
    then
        mess_err:=3;
        return(FALSE);
    endif;
```

```
    Ent3:= ENTITY_T((O_REL2: RELATION((REL1_O: Ent2)
                                      and (:TYPE = "INT")))
                    and (E_S: Sch2));

    if (type = "E")
    then
        tampon:= chemin_EE(Pere,Ent3);
    else
        tampon:= chemin_ER(Ent3,Pere);
    endif;

    if (tampon = TRUE)
    then
        resultat:= TRUE;
    endif;
  endfor;
  if (resultat = FALSE)
  then
      mess_err:= 4;
      return(FALSE);
  else
      return(TRUE);
  endif;
end;
```

B.9 Fonctions RN

Fonction: RN_aff_err

```
procedure RN_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance RN.

Précondition:

     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'N',
     mess_err : numéro du message d'erreur (compris entre 1 et 2).

Postcondition:

     Si mess_err = 1, le message "L'association sélectionnée dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "Une des entités participant à
                      l'association du schéma secondaire n'a pas
                      encore été mise en correspondance" est
                      présenté à l'utilisateur;
}
begin
 if mess_err = 1
 then
   display "L'association sélectionnée dans le schéma secondaire a
            déjà un correspondant";
 else
    display "Une des entités participant à l'association du schéma
             secondaire n'a pas encore été mise en correspondance";
 endif;
end;

procedure RN_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section 5.5.3 pour l'ajout de
     l'association ,sélectionnée dans le schéma secondaire, dans
     le schéma primaire.

Précondition:
```

```
     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'association de Ss à intégrer dans Sp,
     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'N'.

Postcondition:

     MS(Ss,Sp)= MS + RR ObjectInt1*Equivalent*ObjectInt1.
}
begin
   Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
   Lk1:= LINK((L_S: Sch1)
            and (: L_NAME = ObjectInt1));
   Sch2:= SCHEMA(: SCH_NAME = CurSchName2);

   ObjectInt2:= ObjectInt1;
   create Lk2:= LINK((: L_NAME = ObjectInt2)
                     and (: SHORTNAME = SHORTNAME(: Lk1))
                     and (: DATE = DATE(: Lk1))
                     and (L_S: Sch2);
   RR_integ; /* Intégration de LK1 avec Lk2 */
end;
```

Fonction: RN_precond

```
function RN_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'ajout d'une association de Ss dans  Sp sont remplies.
     Ces conditions ont fait l'objet d'une description à la
     section 5.5.3.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     ObjectInt1  : nom de l'association de Ss à intégrer dans Sp,
     TypeObjectInt1 = 'R',
     TypeObjectInt2 = 'N'.

Postcondition:

     RN_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
                        mess_err = 2 si au moins une entité
                                     participant à ObjectInt1 n'a pas
                                     encore été mis en correspondance
                                     avec un objet de Sp;
}
begin
  Sch:= SCHEMA(: SCH_NAME = CurSchName1);
  Lk:= LINK((L_S: Sch)
           and (: L_NAME = ObjectInt1));
  if (RELATION((REL1_O: Lk)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;

  for Ent := ENTITY_T(E_ER: E_ROLE(ER_R: ROLE(R_L: Lk))) do
    if (RELATION((REL1_O: Ent)
                and (: TYPE = "INT"))) = 0
    then
        mess_err:=2;
        return(FALSE);
    endif;
  endfor;
  return(TRUE);
end;
```

B.10 Fonctions AA

Fonction: AA_aff_err

```
procedure AA_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance AA.

Précondition:

     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'A',
     mess_err : numéro du message d'erreur (compris entre 1 et 4).

Postcondition:

     Si mess_err = 1, le message "L'attribut sélectionné dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "Le père direct de l'attribut du
                      schéma secondaire n'a pas encore été mis
                      en correspondance" est présenté à l'utilisateur;
        mess_err = 3, le message "L'attribut sélectionné dans
                      le schéma primaire a déjà été mis en
                      correspondance" est présenté à l'utilisateur;
        mess_err = 4, le message "Il n'y a pas de chemin entre
                      l'attribut du schéma primaire et le
                      correspondant du père de l'attribut du schéma
                      secondaire" est présenté à l'utilisateur.
}
begin
 if mess_err = 1
 then
   display "L'attribut sélectionné dans le schéma secondaire a déjà
            un correspondant";
 else
   if mess_err = 2
   then
     display "Le père direct de l'attribut du schéma secondaire n'a
              pas encore été mis en correspondance";
   else
     if mess_err = 3
     then
      display "L'attribut sélectionné dans le schéma primaire a
               déjà été mis en correspondance";
     else
      display "Il n'y a pas de chemin entre l'attribut du schéma
               primaire et le correspondant du père de l'attribut
               du schéma secondaire";
     endif;
   endif;
 endif;
end;
```

Fonction: AA_integ

```
procedure AA_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section <REFERENCE>(integattribut\VALUE) pour l'intégration de
     l'attribut sélectionné dans le schéma secondaire avec l'attribut
     sélectionné dans le schéma primaire.

Précondition:
```

```
     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'attribut de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'attribut de Sp à intégrer avec ObjectInt1.
     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'A',
     PereObjectInt1 : nom du père de ObjectInt1,
     PereObjectInt2 : nom du père de ObjectInt2,
     Niveau1     : niveau de ObjectInt1 dans la liste des attributs
                   de PereObjectInt1.
     Niveau2     : niveau de ObjectInt2 dans la liste des attributs
                   de PereObjectInt2.

Postcondition:

     MS(Ss,Sp)= MS + AA ObjectInt1*Equivalent*ObjectInt2.
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Type1 := Rch_type_pere(PereObjectInt1, Sch1);
  if (Type1 = "E")
  then
    Pere1:= ENTITY_T(: E_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att1);
  else
    Pere1:= LINK(: L_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att1);
  endif;
  Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
  Type2 := Rch_type_pere(PereObjectInt2, Sch2);
  if (Type2 = "E")
  then
    Pere2:= ENTITY_T(: E_NAME = PereObjectInt2);
    Rch_attribute(ObjectInt2,Niveau2,Pere2,Att2);
  else
    Pere1:= LINK(: L_NAME = PereObjectInt2);
    Rch_attribute(ObjectInt2,Niveau2,Pere2,Att2);
  endif;

  if (MIN_REP(: Att1) < MIN_REP(: Att2))
  then
    modify Att2(: MIN_REP = MIN_REP(: Att1));
  endif;
  if (MAX_REP(: Att1) > MAX_REP(: Att2))
  then
    modify Att2(: MAX_REP = MAX_REP(: Att1));
  endif;

  create rel:= RELATION ((REL1_O: Att1)
                         and (REL2_O: Att2)
                         and (: GROUP_ID = count(RELATION) + 1)
                         and (: TYPE = "INT")
                         and (: TYPE_O1 = "A")
                         and (: TYPE_O2 = "A"));

   if (AT_NAME(: Att1) <> AT_NAME(: Att2))
   then
      if (libelle_unique_attribut(AT_NAME(: Att1), Att2) = TRUE)
      then
          accepte "Voulez-vous intervertir les noms ",resultat;
          if (resultat = OUI)
          then
              modify Att2(:AT_NAME = AT_NAME(: Att1));
          endif;
      endif;
   endif;
end;
```

Fonction: AA_precond

```
function AA_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'intégration d'un attribut de Ss avec un attribut
     de Sp sont remplies. Ces conditions ont fait l'objet d'une
     description à la section 5.5.4.

Précondition:
```

```
     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'attribut de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'attribut de Sp à intégrer avec ObjectInt1.
     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'A',
     PereObjectInt1 : nom du père de ObjectInt1,
     PereObjectInt2 : nom du père de ObjectInt2,
     Niveau1     : niveau de ObjectInt1 dans la liste des attributs
                   de PereObjectInt1.
     Niveau2     : niveau de ObjectInt2 dans la liste des attributs
                   de PereObjectInt2.

Postcondition:

     AA_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
                        mess_err = 2 si le père direct de ObjectInt1
                                     n'a pas encore été mis en
                                     correspondance dans Sp;
                        mess_err = 3 si ObjectInt2 a déjà été mis en
                                     correspondance avec un objet de
                                     Ss;
                        mess_err = 4 s'il existe un chemin entre le
                                     père du correspondant du père
                                     direct de ObjectInt1 et
                                     PereObjectInt2.
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Type1 := Rch_type_pere(PereObjectInt1, Sch1);
  if (Type1 = "E")
  then
    Pere1:= ENTITY_T(: E_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att1);
  else
    Pere1:= LINK(: L_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att1);
  endif;

  if (RELATION((REL1_O: Att1)
               and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;

  Pere_direct:= RECORD(EL_AT: Att1);

  if (RELATION((REL1_O: Pere_direct)
               and (: TYPE = "INT"))) = 0
  then
      mess_err:= 2;
      return(FALSE);
  endif;

  Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
  Type2 := Rch_type_pere(PereObjectInt2, Sch2);
  if (Type2 = "E")
  then
    Pere2:= ENTITY_T(: E_NAME = PereObjectInt2);
    Rch_attribute(ObjectInt2,Niveau2,Pere2,Att2);
  else
    Pere2:= LINK(: L_NAME = PereObjectInt2);
    Rch_attribute(ObjectInt2,Niveau2,Pere2,Att2);
  endif;

  if (RELATION((REL2_O: Att2)
               and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 3;
      return(FALSE);
  endif;

   Corresp:=  RECORD(O_REL2: RELATION((REL1_O: Pere_direct)
                                      (: TYPE = "INT")));
   Type:= TYPE_O2(RELATION(REL2_O: Corresp));
```

```
   case Type of
         'E': case Type2 of
                 'E': resultat:= chemin_EE(Pere2,Corresp);
                  break;
                 'R': resultat:= chemin_ER(Corresp,Pere2);
                 break;
              endcase;
         break;
         'R': case Type2 of
                 'E': resultat:= chemin_ER(Corresp,Pere2);
                  break;
                 'R': resultat:= (L_NAME(:Corresp) = L_NAME(:Pere2));
                 break;
              endcase;
         break;
   endcase;

   if (resultat == FALSE)
   then
        mess_err:= 4;
        return(FALSE);
   else
        return(TRUE);
   endif;
end;
```

B.11 Fonctions AE

Fonction: AE_aff_err

```
procedure AE_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance AE.

Précondition:

     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'E',
     mess_err : numéro du message d'erreur (compris entre 1 et 4).

Postcondition:

     Si mess_err = 1, le message "L'attribut sélectionné dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "Le père direct de l'attribut du
                      schéma secondaire n'a pas encore été mis
                      en correspondance" est présenté à l'utilisateur;
        mess_err = 3, le message "L'entité sélectionnée dans
                      le schéma primaire a déjà été mise en
                      correspondance" est présenté à l'utilisateur;
        mess_err = 4, le message "Il n'y a pas de chemin entre
                      l'attribut du schéma primaire et le
                      correspondant du père de l'attribut du schéma
                      secondaire" est présenté à l'utilisateur.
}
begin
 if mess_err = 1
 then
   display "L'attribut sélectionné dans le schéma secondaire a déjà
            un correspondant";
 else
   if mess_err = 2
   then
     display "Le père direct de l'attribut du schéma secondaire n'a
              pas encore été mis en correspondance";
   else
     if mess_err = 3
     then
      display "L'entité sélectionnée dans le schéma primaire a
               déjà été mise en correspondance";
     else
```

```
      display "Il n'y a pas de chemin entre l'attribut du schéma
               primaire et le correspondant du père de l'attribut
               du schéma secondaire";
    endif;
  endif;
 endif;
end;
```

Fonction: AE_Integ

```
procedure AE_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section <REFERENCE>(integattribut\VALUE) pour l'intégration de
     l'attribut sélectionné dans le schéma secondaire avec l'entité
     sélectionnée dans le schéma primaire.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'attribut de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'entité de Sp à intégrer avec ObjectInt1.
     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'E',
     PereObjectInt1 : nom du père de ObjectInt1,
     Niveau1     : niveau de ObjectInt1 dans la liste des attributs
                   de PereObjectInt1.

Postcondition:

     MS(Ss,Sp)= MS + AE ObjectInt1*Equivalent*ObjectInt2.
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Type1 := Rch_type_pere(PereObjectInt1, Sch1);
  if (Type1 = "E")
  then
    Pere1:= ENTITY_T(: E_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att);
  else
    Pere1:= LINK(: L_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att);
  endif;

  Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
  Ent:= ENTITY_T((:E_NAME = ObjectInt2)
                 and (E_S: Sch2);

  create rel:= RELATION ((REL1_O: Att)
                         and (REL2_O: Ent)
                         and (: GROUP_ID = count(RELATION) + 1)
                         and (: TYPE = "INT")
                         and (: TYPE_O1 = "A")
                         and (: TYPE_O2 = "E"));

   if (AT_NAME(: Att) <> E_NAME(: Ent))
   then
      if (libelle_unique(AT_NAME(: Att) = TRUE)
      then
          accepte "Voulez-vous intervertir les noms ",resultat;
          if (resultat = OUI)
          then
              modify Ent(:E_NAME = AT_NAME(: Att));
          endif;
      endif;
   endif;

end;
```

Fonction: AE_precond

```
function AE_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'intégration d'un attribut de Ss avec une entité
     de Sp sont remplies. Ces conditions ont fait l'objet d'une
     description à la section 5.5.4.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'attribut de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'entité de Sp à intégrer avec ObjectInt1.
     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'E',
     PereObjectInt1 : nom du père de ObjectInt1,
     Niveau1     : niveau de ObjectInt1 dans la liste des attributs
                   de PereObjectInt1.

Postcondition:

     AE_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
                        mess_err = 2 si le père direct de ObjectInt1
                                     n'a pas encore été mis en
                                     correspondance dans Sp;
                        mess_err = 3 si ObjectInt2 a déjà été mis en
                                     correspondance avec un objet de
                                     Ss;
                        mess_err = 4 s'il existe un chemin entre le
                                     père du correspondant du père
                                     direct de ObjectInt1 et
                                     PereObjectInt2.
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Type1 := Rch_type_pere(PereObjectInt1, Sch1);
  if (Type1 = "E")
  then
    Pere1:= ENTITY_T(: E_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att);
  else
    Pere1:= LINK(: L_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att);
  endif;

  if (RELATION((REL1_O: Att)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;

  Pere_direct:= RECORD(EL_AT: Att);

  if (RELATION((REL1_O: Pere_direct)
              and (: TYPE = "INT"))) = 0
  then
      mess_err:= 2;
      return(FALSE);
  endif;

  Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
  Ent:= ENTITY_T((E_S: Sch2)
                 and (: E_NAME = ObjectInt2));
  if (RELATION((REL2_O: Ent)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 3;
      return(FALSE);
  endif;
```

```
   Corresp:=  RECORD(O_REL2: RELATION((REL1_O: Pere_direct)
                                      (: TYPE = "INT")));
   Type:= TYPE_O2(RELATION(REL2_O: Corresp));

   if (Type = 'E')
   then
       resultat:= chemin_EE(Ent,Corresp);
   else
       resultat:= chemin_ER(Ent,Corresp);
   endif;

   if (resultat == FALSE)
   then
        mess_err:= 4;
        return(FALSE);
   else
        return(TRUE);
   endif;
end;
```

B.12 Fonctions AR

Fonction: AR_aff_err

```
procedure AR_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance AR.

Précondition:

     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'R',
     mess_err : numéro du message d'erreur (compris entre 1 et 4).

Postcondition:

     Si mess_err = 1, le message "L'attribut sélectionné dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "Le père direct de l'attribut du
                      schéma secondaire n'a pas encore été mis
                      en correspondance" est présenté à l'utilisateur;
        mess_err = 3, le message "L'association sélectionnée dans
                      le schéma primaire a déjà été mise en
                      correspondance" est présenté à l'utilisateur;
        mess_err = 4, le message "Il n'y a pas de chemin entre
                      l'attribut du schéma primaire et le
                      correspondant du père de l'attribut du schéma
                      secondaire" est présenté à l'utilisateur.
}
begin
 if mess_err = 1
 then
   display "L'attribut sélectionné dans le schéma secondaire a déjà
            un correspondant";
 else
   if mess_err = 2
   then
     display "Le père direct de l'attribut du schéma secondaire n'a
              pas encore été mis en correspondance";
   else
     if mess_err = 3
     then
      display "L'association sélectionnée dans le schéma primaire a
               déjà été mise en correspondance";
     else
      display "Il n'y a pas de chemin entre l'attribut du schéma
               primaire et le correspondant du père de l'attribut
               du schéma secondaire";
     endif;
   endif;
 endif;
```

```
end;
```

Fonction: AR_Integ

```
procedure AR_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section <REFERENCE>(integattribut\VALUE) pour l'intégration de
     l'attribut sélectionné dans le schéma secondaire avec
     l'asssociation sélectionnée dans le schéma primaire.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'attribut de Ss à intégrer dans Sp,
     ObjectInt2  : nom de l'association de Sp à intégrer avec
                   ObjectInt1.
     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'R',
     PereObjectInt1 : nom du père de ObjectInt1,
     Niveau1     : niveau de ObjectInt1 dans la liste des attributs
                   de PereObjectInt1.

Postcondition:

     MS(Ss,Sp)= MS + AR ObjectInt1*Equivalent*ObjectInt2.
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Type1 := Rch_type_pere(PereObjectInt1, Sch1);
  if (Type1 = "E")
  then
    Pere1:= ENTITY_T(: E_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att);
  else
    Pere1:= LINK(: L_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att);
  endif;

  Sch2:= SCHEMA(: SCH_NAME = CurSchName2);
  Lk:= LINK((:L_NAME = ObjectInt2)
                and (L_S: Sch2);

  create rel:= RELATION ((REL1_O: Att)
                         and (REL2_O: Lk)
                         and (: GROUP_ID = count(RELATION) + 1)
                         and (: TYPE = "INT")
                         and (: TYPE_O1 = "A")
                         and (: TYPE_O2 = "R"));

   if (AT_NAME(: Att) <> R_NAME(: Lk))
   then
      if (libelle_unique(AT_NAME(: Att) = TRUE)
      then
          accepte "Voulez-vous intervertir les noms ",resultat;
          if (resultat = OUI)
          then
              modify Lk(:L_NAME = AT_NAME(: Att));
          endif;
      endif;
   endif;
end;
```

Fonction: AR_precond

```
function AR_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'intégration d'un attribut de Ss avec une association
     de Sp sont remplies. Ces conditions ont fait l'objet d'une
     description à la section 5.5.4.

Précondition:
```

```
    CurSchName1 : nom du schéma secondaire (Ss),
    CurSchName2 : nom du schéma primaire (Sp),
    ObjectInt1  : nom de l'attribut de Ss à intégrer dans Sp,
    ObjectInt2  : nom de l'association de Sp à intégrer avec
                  ObjectInt1.
    TypeObjectInt1 = 'A',
    TypeObjectInt2 = 'R',
    PereObjectInt1 : nom du père de ObjectInt1,
    Niveau1     : niveau de ObjectInt1 dans la liste des attributs
                  de PereObjectInt1.

Postcondition:

    AR_precond = TRUE  si toutes les conditions sont remplies;
                       mess_err = 0.
                 FALSE si au moins une des conditions n'est pas
                       remplie;
                       mess_err = 1 si ObjectInt1 a déjà un
                                    correspondant dans Sp;
                       mess_err = 2 si le père direct de ObjectInt1
                                    n'a pas encore été mis en
                                    correspondance dans Sp;
                       mess_err = 3 si ObjectInt2 a déjà été mis en
                                    correspondance avec un objet de
                                    Ss;
                       mess_err = 4 s'il existe un chemin entre le
                                    père du correspondant du père
                                    direct de ObjectInt1 et
                                    PereObjectInt2.
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Type1 := Rch_type_pere(PereObjectInt1, Sch1);
  if (Type1 = "E")
  then
    Pere1:= ENTITY_T(: E_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att);
  else
    Pere1:= LINK(: L_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att);
  endif;

  if (RELATION((REL1_O: Att)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;

  Pere_direct:= RECORD(EL_AT: Att);

  if (RELATION((REL1_O: Pere_direct)
              and (: TYPE = "INT"))) = 0
  then
      mess_err:= 2;
      return(FALSE);
  endif;

  Sch2:= SCHEMA(:SCH_NAME = CurSchName2);
  Lk:= LINK((L_S: Sch2)
           and (: L_NAME = ObjectInt2));
  if (RELATION((REL2_O: Lk)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 3;
      return(FALSE);
  endif;

   if (Type = 'E')
   then
       resultat:= chemin_ER(Corresp,Lk);
   else
       resultat:= (L_NAME(: Corresp) = L_NAME(: Lk));
   endif;

   if (resultat == FALSE)
   then
        mess_err:= 4;
        return(FALSE);
   else
        return(TRUE);
   endif;
```

```
 return(TRUE);
end;
```

B.13 Fonctions AN

Fonction: AN_aff_err

```
procedure AN_aff_err ( mess_err : integer): boolean;
{
Spécification:

But: Cette procédure affiche le message d'erreur en rapport
     avec la condition qui n'a pas été vérifiée pour
     le type de correspondance AN.

Précondition:

     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'N',
     mess_err : numéro du message d'erreur (compris entre 1 et 2).

Postcondition:

     Si mess_err = 1, le message "L'attribut sélectionné dans
                      le schéma secondaire a déjà un correspondant"
                      est présenté à l'utilisateur;
        mess_err = 2, le message "Le père direct de l'attribut du
                      schéma secondaire n'a pas encore été mis
                      en correspondance" est présenté à l'utilisateur;
}
begin
 if mess_err = 1
 then
   display "L'attribut sélectionné dans le schéma secondaire a déjà
            un correspondant";
 else
   display "Le père direct de l'attribut du schéma secondaire n'a
            pas encore été mis en correspondance";
 endif;
end;
```

Fonction: AN_integ

```
procedure AN_integ;
{
Spécification:

But: Cette procédure réalise les actions spécifiées à la
     section <REFERENCE>(integattribut\VALUE) pour l'ajout de
     l'attribut ,sélectionné dans le schéma secondaire, dans
     le schéma primaire.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     CurSchName2 : nom du schéma primaire (Sp),
     ObjectInt1  : nom de l'attribut de Ss à intégrer dans Sp,
     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'N',
     PereObjectInt1 : nom du père de ObjectInt1,
     Niveau1      : niveau de ObjectInt1 dans la liste des attributs
                    de PereObjectInt1.

Postcondition:
```

```
    MS(Ss,Sp)= MS + AA ObjectInt1*Equivalent*ObjectInt1.
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Type1 := Rch_type_pere(PereObjectInt1, Sch1);
  if (Type1 = "E")
  then
    Pere1:= ENTITY_T(: E_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att);
  else
    Pere1:= LINK(: L_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att);
  endif;

  Pere_direct:= RECORD(EL_AT: Att);
  Corresp:= RECORD(O_REL2: RELATION(REL1_O: Pere_direct);

  create Att2:= ((: AT_NAME = AT_NAME(: Att1))
                 and (: SHORT_NAME = SHORT_NAME(: Att1))
                 and (: DATE = DATE(: Att1))
                 and (: MIN_REP = MIN_REP(: Att1))
                 and (: MAX_REP = MAX_REP(: Att1))
                 and (AT_EL: Corresp));

  AA_integ; /* intégration de Att1 avec Att2 */

end;
```

Fonction: AN_precond

```
function AN_precond (var mess_err : integer): boolean;
{
Spécification:

But: Cette fonction vérifie que les conditions préalables à
     l'ajout d'un attribut de Ss dans Sp sont remplies. Ces
     conditions ont fait l'objet d'une description à la
     section 5.5.4.

Précondition:

     CurSchName1 : nom du schéma secondaire (Ss),
     ObjectInt1  : nom de l'attribut de Ss à intégrer dans Sp,
     TypeObjectInt1 = 'A',
     TypeObjectInt2 = 'N',
     PereObjectInt1 : nom du père de ObjectInt1,
     Niveau1     : niveau de ObjectInt1 dans la liste des attributs
                   de PereObjectInt1.

Postcondition:

     AN_precond = TRUE  si toutes les conditions sont remplies;
                        mess_err = 0.
                  FALSE si au moins une des conditions n'est pas
                        remplie;
                        mess_err = 1 si ObjectInt1 a déjà un
                                     correspondant dans Sp;
                        mess_err = 2 si le père direct de ObjectInt1
                                     n'a pas encore été mis en
                                     correspondance dans Sp;
}
begin
  Sch1:= SCHEMA(: SCH_NAME = CurSchName1);
  Type1 := Rch_type_pere(PereObjectInt1, Sch1);
  if (Type1 = "E")
  then
    Pere1:= ENTITY_T(: E_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att1);
  else
    Pere1:= LINK(: L_NAME = PereObjectInt1);
    Rch_attribute(ObjectInt1,Niveau1,Pere1,Att1);
  endif;

  if (RELATION((REL1_O: Att1)
              and (: TYPE = "INT"))) <> 0
  then
      mess_err:= 1;
      return(FALSE);
  endif;

  Pere_direct:= RECORD(EL_AT: Att1);
```

```
  if (RELATION((REL1_O: Pere_direct)
              and (: TYPE = "INT"))) = 0
  then
      mess_err:= 2;
      return(FALSE);
  endif;

 return(TRUE);
end;
```

B.14 Fonctions complémentaires

Fonction: chemin_EE

```
function chemin_EE ( Ent1, Ent2 : ENTITY_T): boolean;
{
Spécification:

But: Cette fonction vérifier s'il existe une association qui relie les
     deux entités Ent1 et Ent2.

Précondition:

     CurSchName2   : nom du schéma primaire;
     Ent1 : une entité du schéma primaire;
     Ent2 : une entité du schéma primaire;

Postcondition:

     chemin_EE = TRUE s'il existe une association dans le schéma primaire
                      que relie les deux entités Ent1 et Ent2;
               = FALSE sinon.
}
begin
  Sch:=SCHEMA(:SCH_NAME = CurSchName2);
  if (E_NAME(:Ent1) = E_NAME(:Ent2))
  then
      return(TRUE);
  endif;

  Trouve:= FALSE;
  Lk:= first(LINK((L_R: ROLE(R_ER: E_ROLE(ER_E: Ent1)))
                  and (L_S: Sch)));
  while (Lk <> ()) and (Trouve = FALSE) do
    Ent3:= first(ENTITY_T((E_ER: E_ROLE(ER_R: ROLE(R_L: Lk)))
                          and (E_S: Sch)));
    while (Ent3 <> ()) and (Trouve = FALSE) do
       if (E_NAME(:Ent3) = E_NAME(:Ent2))
       then
           Trouve:= TRUE;
       else
           Ent3:= next(ENTITY_T((E_ER: E_ROLE(ER_R: ROLE(R_L: Lk)))
                                and (E_S: Sch)));
       endif;
    endwhile;
    Lk:= next(LINK((L_R: ROLE(R_ER: E_ROLE(ER_E: Ent1)))
                   and (L_S: Sch)));
   endwhile;
   return(Trouve)
end;
```

Fonction: chemin_ER

```
function chemin_ER ( Ent : ENTITY_T, Lk : LINK): boolean;
{
Spécification:

But: Cette fonction vérifie si l'entité Ent participe à
     l'association Lk dans le schéma primaire.

Précondition:

     CurSchName2 : nom du schéma primaire;
     Ent : une entité du schéma primaire;
     Lk  : une association du schéma primaire;

Postcondition:

     chemin_ER = TRUE si l'entité Ent joue l'association Lk.
               = FALSE sinon.
}
begin
  Sch:= SCHEMA(: SCH_NAME = CurSchName2);
  Trouve:= FALSE;
  Lk1:= first(LINK((L_R: ROLE(R_ER: E_ROLE(ER_E: Ent)))
                  and (L_S: Sch)));
  while (Lk <> ()) and (Trouve = FALSE) do
    if (L_NAME(:Lk) = L_NAME(:Lk1))
       then
           Trouve:= TRUE;
       else
           Lk1:= next(LINK((L_R: ROLE(R_ER: E_ROLE(ER_E: Ent)))
                           and (L_S: Sch)));
       endif;
  endwhile;
  return(Trouve)
end;
```

Fonction: libelle_unique

```
function libelle_unique( Libelle : STRING[31]): boolean;
{
Spécification:

But: Cette fonction vérifie si le libelle LIBELLE est unique parmi
     les entités (E_NAME) et les associations (L_NAME) du schéma
     primaire.

Précondition:

     CurSchName2  : Nom du schéma primaire;
     Libelle      : libellé d'un objet du schéma secondaire que
                    l'on désire intégrer dans le schéma primaire;

Postcondition:
```

```
     libelle_unique = TRUE si le libelle LIBELLE est unique parmi
                           les entités et les associations du schéma
                           primaire;
                    = FALSE sinon.
}
begin
  Sch:= (SCHEMA: CurSchName2);
  Ent:= first(ENTITY_T(E_S: Sch));
  while Ent <> () do
    if (E_NAME(:Ent) = libelle)
    then
        return(FALSE);
    endif
    Ent:= next(ENTITY_T(E_S: Sch));
  endwhile;

  Lk:= first(LINK(L_S: Sch));
  while Lk <> () do
    if (L_NAME(:Lk) = libelle)
    then
        return(FALSE);
    endif
  Lk:= next(LINK(L_S: Sch));
  endwhile;
  return(TRUE);
end;
```

Fonction: libelle_unique_attribut

```
function libelle_unique_attribut( Libelle : STRING[31],
                                  Att : ATTRIBUTE): boolean;
{
Spécification:

But: Cette fonction vérifie si le libelle LIBELLE est unique parmi
     les attributs de même niveau que ATT.

Précondition:

     Libelle      : libellé de l'attribut du schéma secondaire que
                    l'on désire intégrer dans le schéma primaire;
     Att          : l'attribut du schéma primaire mis en
                    correspondance avec l'attribut de libelle LIBELLE
                    du schéma secondaire.

Postcondition:

     libelle_unique_attribut = TRUE si le libelle LIBELLE est unique parmi
                                   les attributs du même niveau que ATT.
                             = FALSE sinon.
}
begin
  Pere:= RECORD(EL_AT: Att);
  Att1:= first(ATTRIBUTE(AT_EL: Pere));
  while Att1 <> () do
   if (AT_NAME(:Att1) = libelle)
   then
       return(FALSE);
   endif;

   Att1:= next(ATTRIBUTE(AT_EL: Pere));
  endwhile;
  return(TRUE);
end;
```

Fonction: Rch_attribute

```
function Rch_attribute ( nom_obj : string[31]; niv : integer;
                         pere : OBJECT; var Att : ATTRIBUTE);
{
Spécification:

But: Cette Procédure (récursive) recherche un attribut de nom NOM_OBJ,
     appartenant à PERE et étant de niveau NIV.

Précondition:

     nom_obj : nom d'un objet du schéma Sch,
     niv     : niveau de l'objet,
     pere    : le père de  l'objet nom_obj.

Postcondition:

    Att  est l'attribut de nom NOM_OBJ appartenant à PERE et étant de
         niveau NIV.
}
begin
  Att:= first(ATTRIBUTE(EL_AT: pere);
  while Att <> () do
    if (niv > 1)
    then
       niv:= niv - 1;
       Rch_attribute(nom_obj,niv,pere,Att);
    else
       if (AT_NAME(:Att) = nom_obj)
       then
          return;
       else
          Att:= next(ATTRIBUTE(EL_AT: pere);
       endif;
    endif;
  endwhile;
end;
```

Fonction: Rch_type_pere

```
function Rch_type_pere ( nom_pere : string[31]; Sch : SCHEMA): char;
{
Spécification:

But: Cette fonction cherche le type de l'objet de nom NOM_PERE et appartenant
     au schéma SCH.

Précondition:

     Sch      : un schéma,
     nom_pere : nom d'un objet du schéma Sch.

Postcondition:

     Rch_type_pere = 'E' si nom_pere est une entité dans Sch;
                   = 'R' si nom_pere est une association dans Sch.
}
begin
  for Ent := ENTITY_T do
      if (E_NAME(:Ent) = nom_pere)
      then
        return("E");
      endif;
  endfor;
```

```
  for Lk := LINK do
      if (L_NAME(:Lk) = nom_pere)
      then
        return("R");
      endif;
  endfor;
  return(TRUE);
end;
```

ANNEXE C

CODE SOURCE DU MODULE D'INTÉGRATION

Le code source du module d'intégration que nous avons réalisé dans l'atelier logiciel TRAMIS n'a été reproduit qu'en un seul exemplaire car il fait partie d'un produit commercialisé.

Le lecteur intéressé par le contenu de cette partie devra demander l'autorisation de consulter cette annexe auprès de monsieur Jean-Luc HAINAUT, professeur aux Facultés Universitaires N-D de la Paix de Namur, qui possède l'unique exemplaire.

ANNEXE D

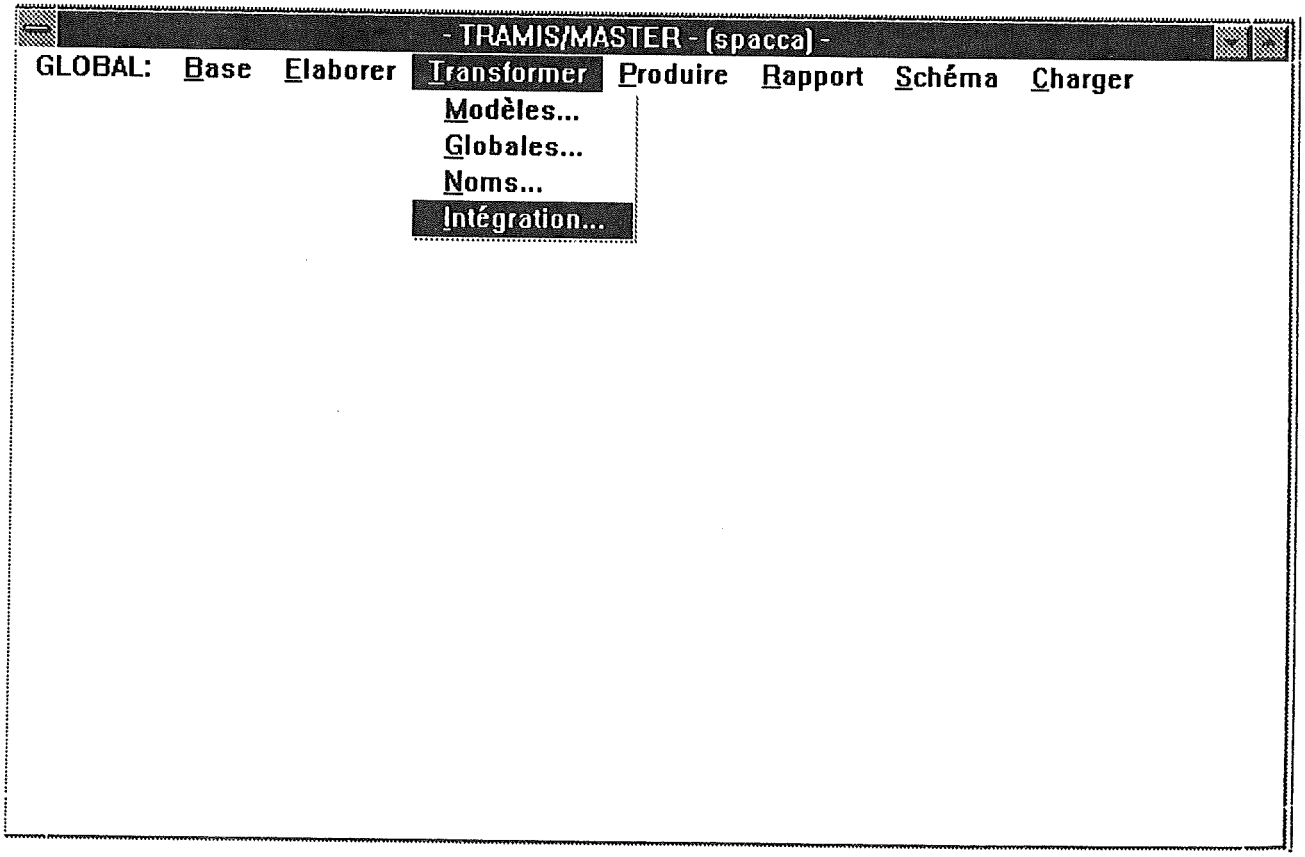
SESSION D'INTÉGRATION DANS TRAMIS

D.1 Exemple de session d'intégration dans TRAMIS

Dans cette annexe, nous présentons un exemple de session d'intégration en reprenant l'exemple **Propriétaire de voiture** de la section 5.7. Pour cette illustration, nous utilisons une base de données que nous avons définie dans TRAMIS et reprenant la plupart des exemples illustrés dans *[Spaccapietra & Parent 90]*.

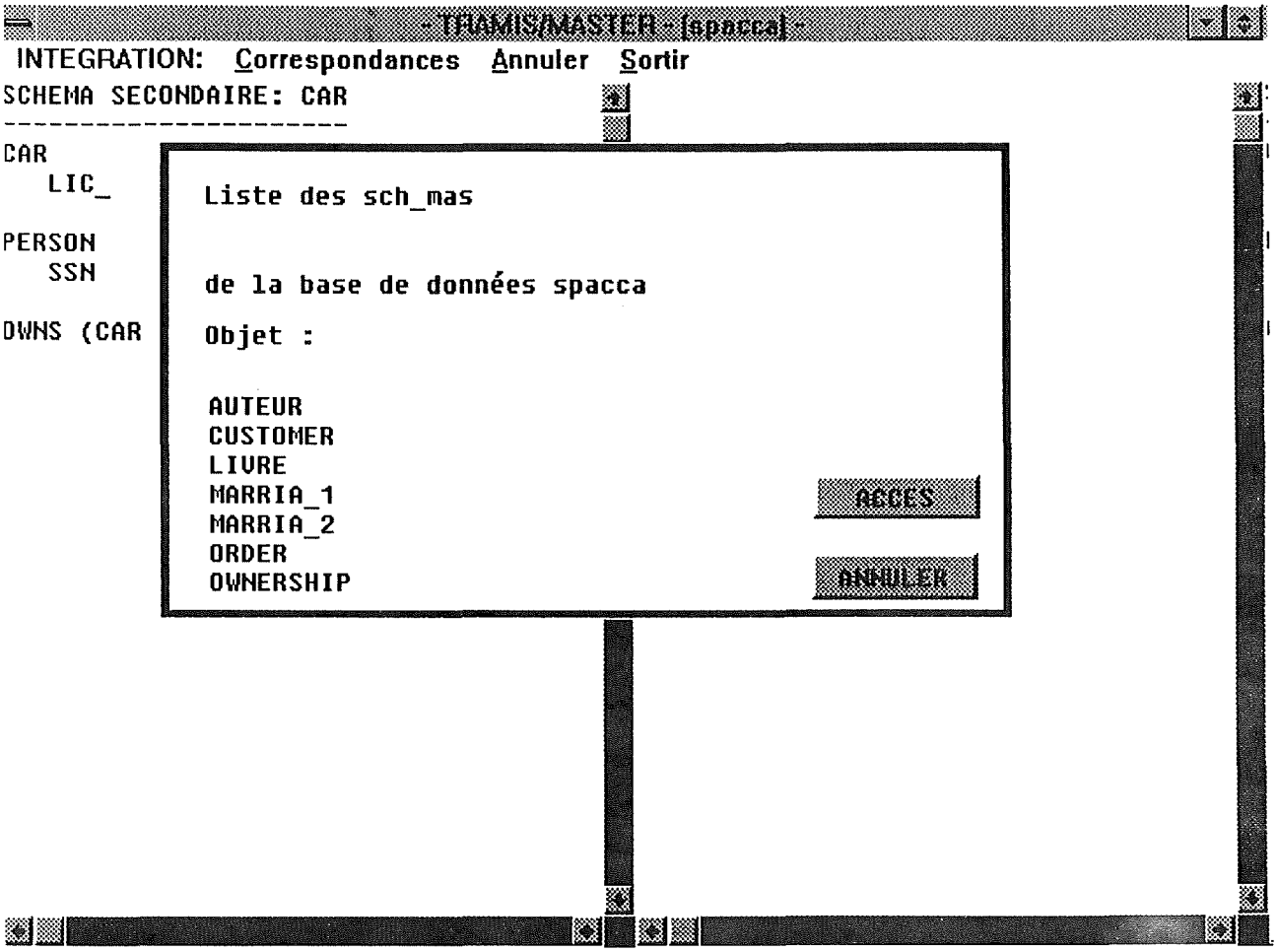
Une session à l'intérieur de l'atelier logiciel TRAMIS commence toujours par l'ouverture d'une base de données. Nous ouvrons pour notre exemple la base de données "spacca.db" qui contient les exemples que nous utiliserons pour cette session. Nous pouvons voir à la Figure D–1 le menu qui permet de lancer le processus d'intégration.

Figure D–1: Début du processus d'intégration



Une fois le processus d'intégration lancé, celui-ci demande le nom des schémas à intégrer (le schéma secondaire puis le schéma primaire). Nous choisissons parmi les schémas de la base de données, le schéma secondaire "CAR" et le schéma primaire "OWNERSHIP" (voir Figure D–2). Les schémas choisis seront affichés dans deux fenêtres situées côte à côte.

Figure D–2: Choix du schéma primaire



A ce moment, nous pouvons commencer l'intégration proprement dite par la mise en correspondance des éléments du schéma secondaire avec un objet du schéma primaire. Nous décidons de commencer par intégrer l'entité "CAR". Après avoir sélectionné la ligne où se trouve l'objet "CAR" dans le schéma secondaire (sur la partie gauche de l'écran), nous voyons la boîte de dialogue de la Figure D–3 se superposer aux fenêtres affichées pour demander s'il existe un correspondant. Nous répondons à cette interrogation par l'affirmative en cliquant sur le bouton "Yes".

Figure D–3: Demande du correspondant

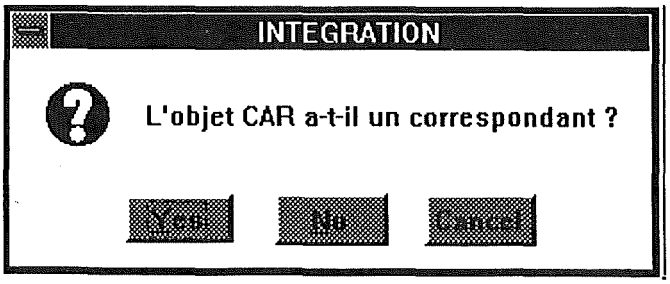
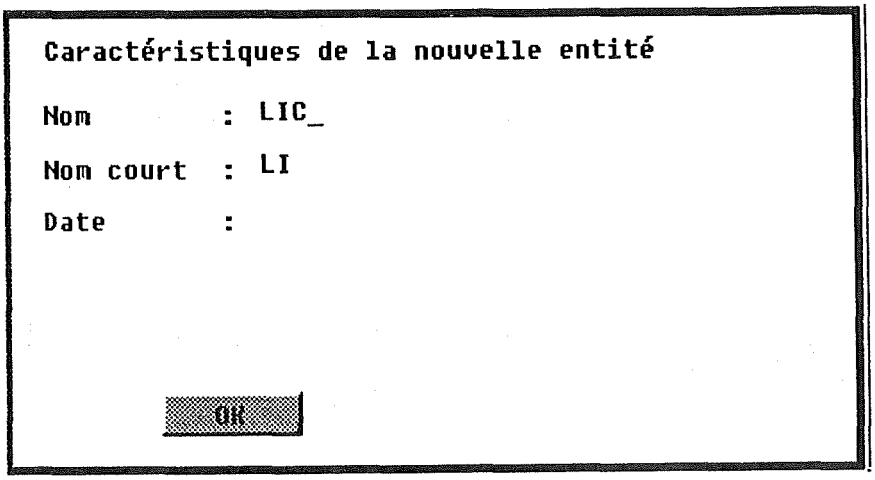


Figure D–4: Demande du nom



Nous sélectionnons dans le schéma primaire (le schéma à droite de l'écran) la ligne où se trouve le correspondant. Il s'agit dans notre exemple de l'attribut "LIC_". Nous avons une correspondance mixte (ENTITE/ATTRIBUT) qui vérifie les préconditions de la fonction EA et à pour résultat la transformation de l'attribut "LIC_" en entité. Lors de cette transformation, l'outil affiche une boîte de dialogue (voir Figure D–4) pour demander des informations sur cette nouvelle entité.

Après quoi, la possibilité de remplacer le nom "LIC_" par "CAR" dans le schéma primaire nous est offerte (voir Figure D–5), puisque le nom "CAR" n'est pas présent parmis les entités ni les associations du schéma primaire. En acceptant le changement, nous obtenons le résultat de la Figure D–6.

Une aide nous est donnée, en sélectionnant le menu "Correspondances", par la visualisation des correspondances déjà introduites (voir Figure D–7). Nous avons le choix d'afficher ces correspondances dans les deux sens (Primaire vers secondaire ou vice versa). Nous prenons le sens du schéma secondaire vers le schéma primaire et obtenons la confirmation de ce que

Figure D–5: Demande de permutation des nom

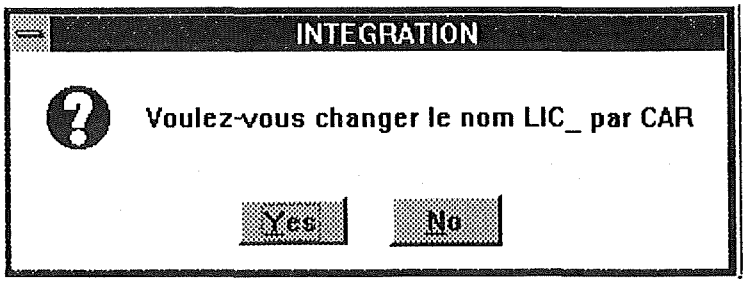
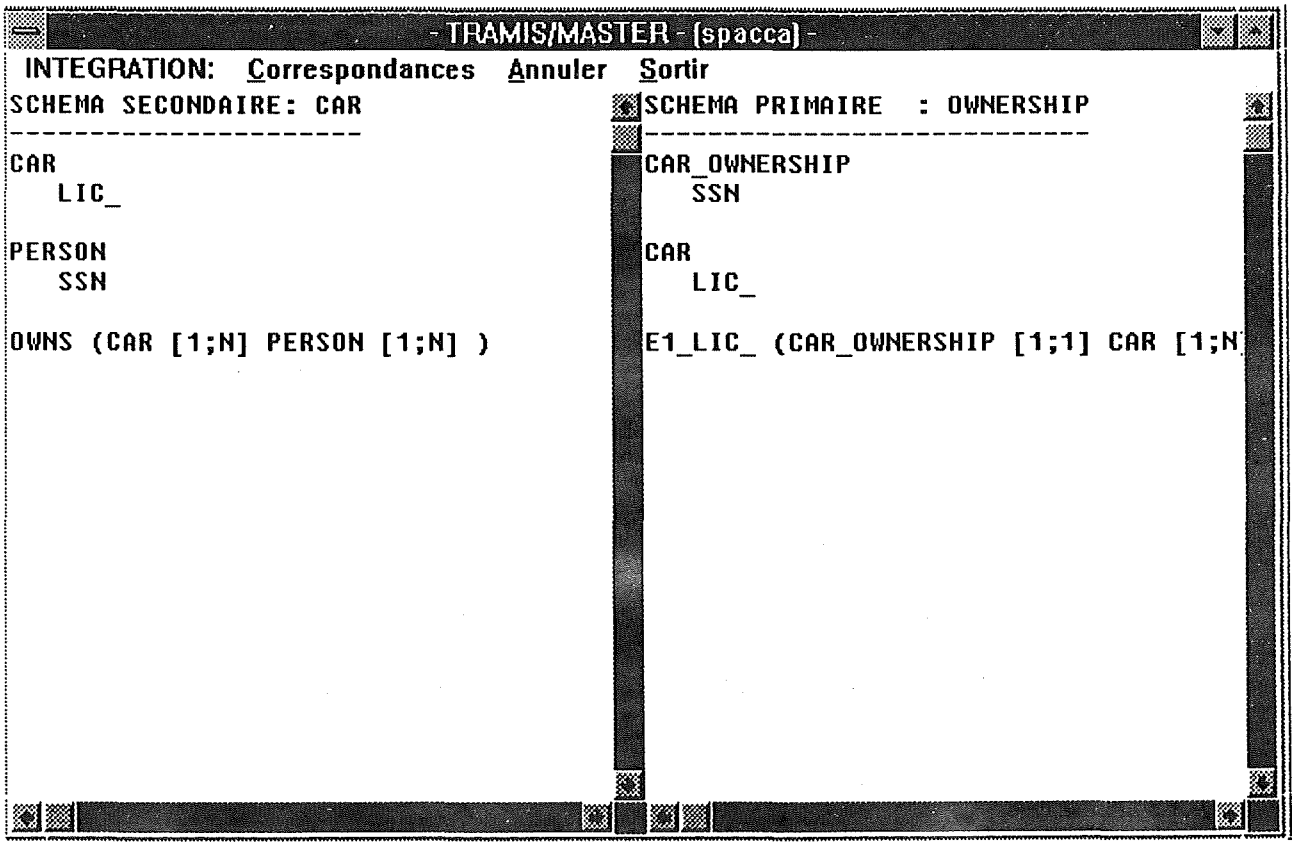


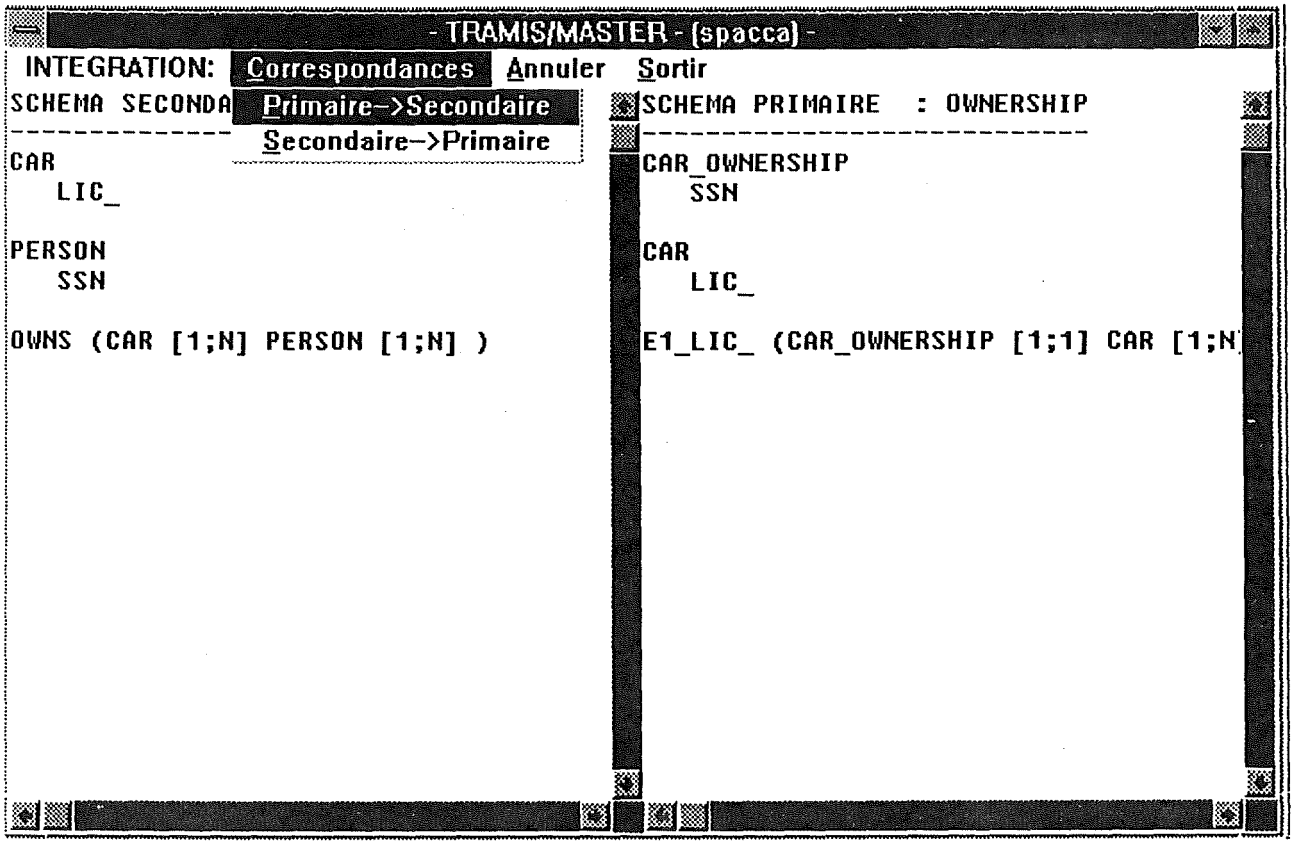
Figure D–6: Résultat après la première correspondance



nous attendions à la Figure D–8, à savoir l'existance de la correspondance entre l'entité du schéma secondaire et l'attribut (qui entre temps est devenu une entité) du schéma primaire.

Après être sorti de la fenêtre des correspondances, par le menu du même nom (voir Figure D–8), l'utilisateur introduit ensuite les correspondances:

Figure D–7: Comment demander à voir les correspondance




```
CS : AA Lic_.Car*Equivalent*Lic_.Car
CS : EA Person*Equivalent*Ssn.Car_ownership
CS : AA Ssn.Person*Equivalent*Ssn.Person
CS : RE Owns*Equivalent*Car_ownership
```

Nous obtenons, après permutation du nom de l'entité "SSN" du schéma primaire avec le nom de l'entité "PERSON", le schéma final illustré à la Figure D–9 avec pour mapping structurel le résultat de la Figure D–10.

Cet exemple que nous venons de présenter ne contient délibérément aucune erreur (de manipulation, de choix des correspondances, . . .) de notre part. Cependant, l'outil que nous avons implémenté dans l'atelier logiciel TRAMIS gère les erreurs qui pourrait survenir lors d'une session si on ne respecte pas les préconditions ou si on fait une mauvaise manipulation.

Figure D–8: La correspondance introduite

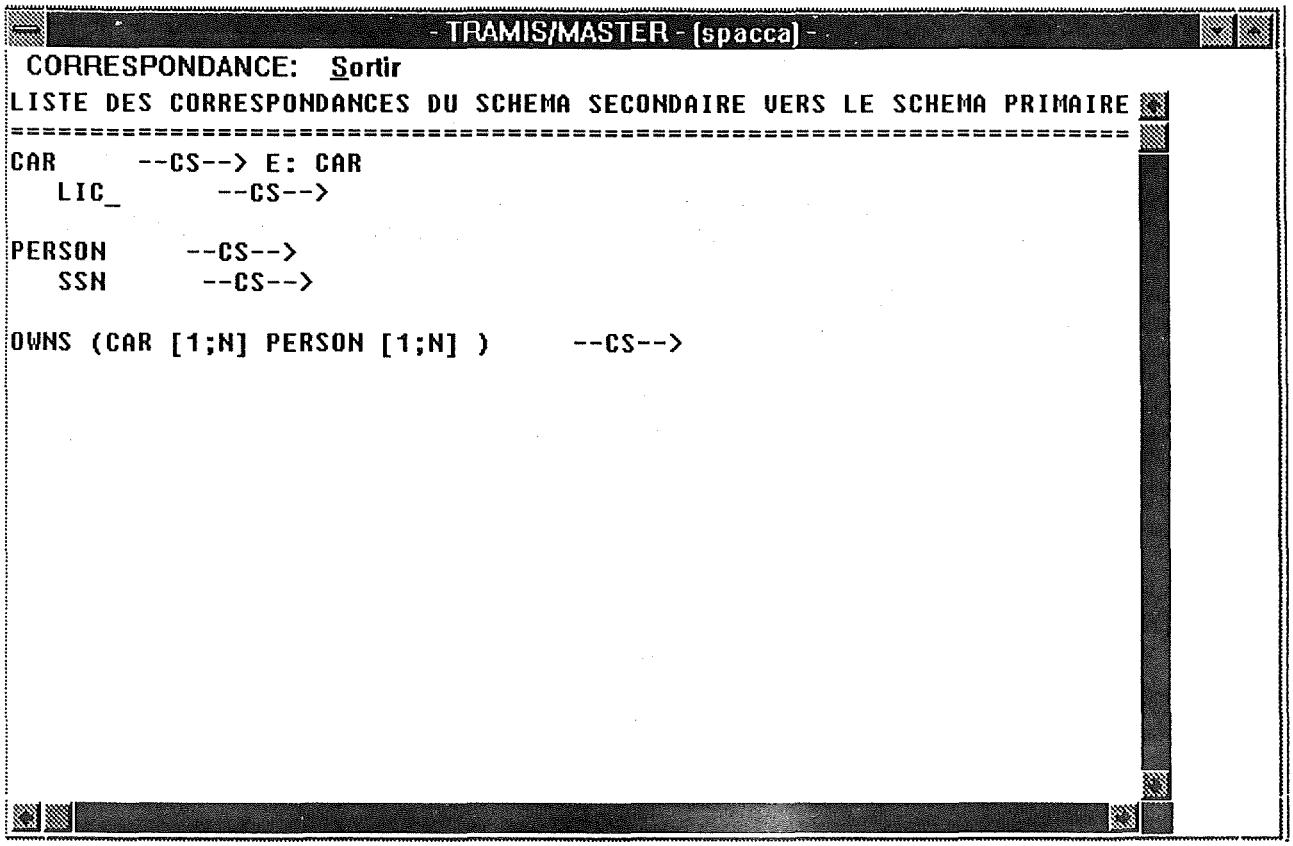


Figure D–9: Résultat final

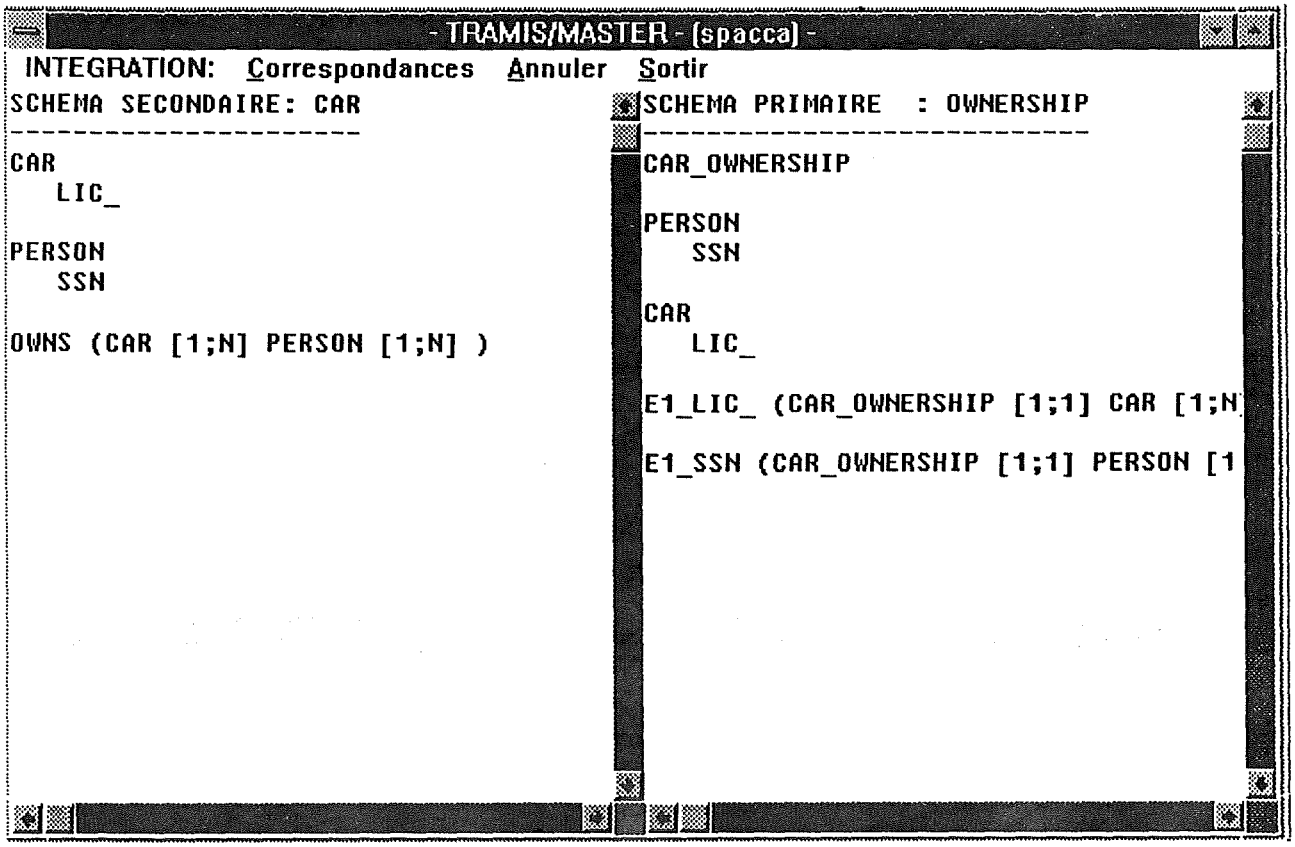
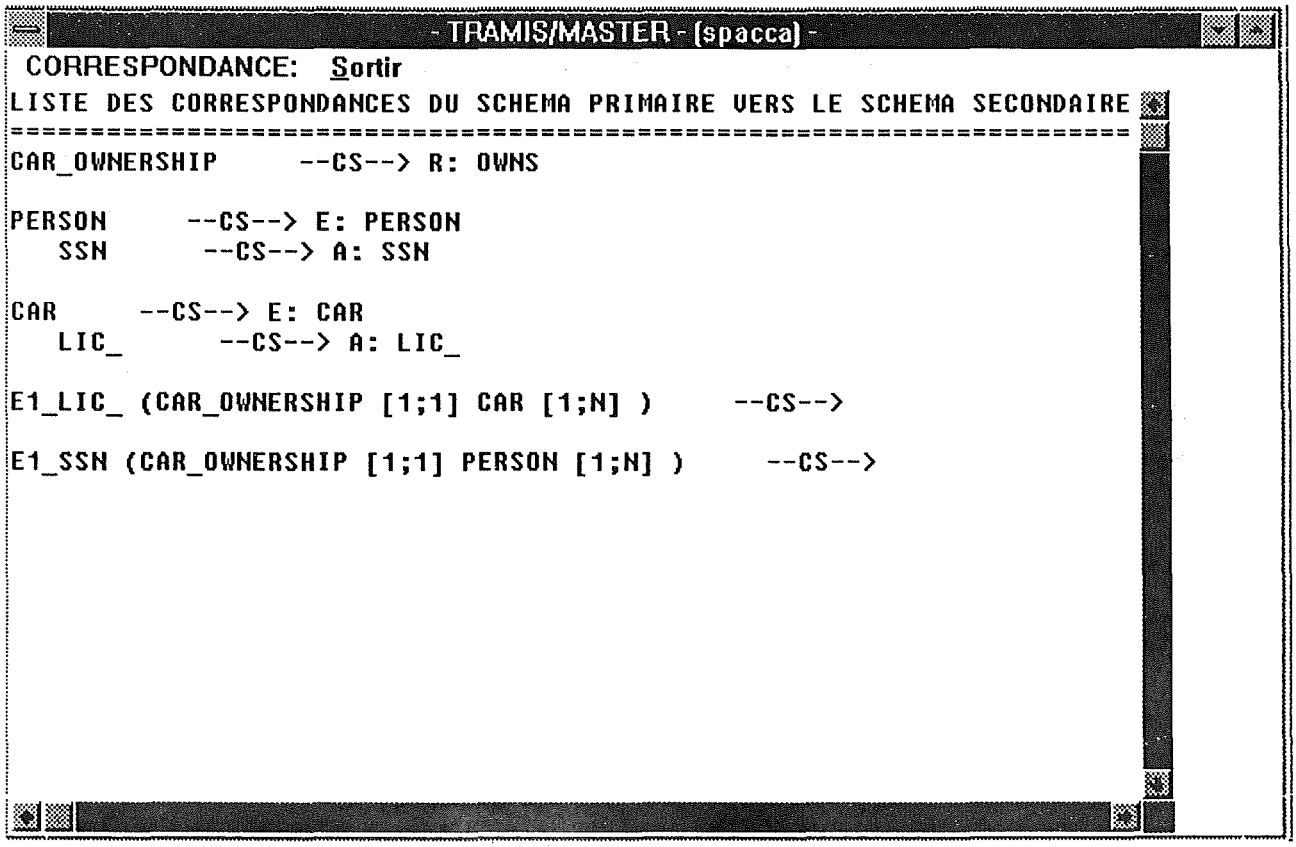


Figure D–10: La liste des correspondances introduites



BIBLIOGRAPHIE

[ANSI 75]: ANSI/X3/SPARC study group on database management report. Tsichritzis ed. Fev 1975 réimprimé dans Information systems, vol 3, 1978.

[Batini & Lenzerini 84]: BATINI, C., AND LENZERINI, M. Nov. 1984. "A methodology for data schema integration in the entity relationship model". IEEE Trans. Softw. Eng. SE-10, 6 , 650-663.

[Batini & Al 86]: BATINI, C., LENZERINI, M., NAVATHE, S.B. Dec 1986. "A comparative analysis of methodologies for database schema integration". ACM computing survey, Vol 18, N°4.

[Belfar 84]: BELFAR, K. 1984. "Méthode d'intégration de schémas et application aux schémas exprimés dans un modèle de type E-R-A". Thèse de doctorat, Université Pierre et Marie Curie (PARIS VI).

[Biskup & Convent 86]: BISKUP, J., CONVENT, B. 1987. "A formal view integration method". ACM-SIGMOD.

[Bodart & Pigneur 89]: BODART, F., PIGNEUR, Y. 1989. "Conception assistée des systèmes d'information. Méthode - Modèle - Outils". 2° édition. Edition Masson.

[Bouzeghoub 84]: BOUZEGHOUB, M. 1984. "MORSE: A functional query language and its semantic data model". Proc of 84 Trends and Application conf. on Databases, IEEE-NBS Gaithersburg (USA).

[Bouzeghoub & Al 90]: BOUZEGHOUB, M., COMYN-WATTIAU, I. October 8-10 1990. "View integration by semantic unification and transformation of data structures", 9th international conference on entity-relationship, Lausanne(ch).

[Brès 90]: BRÈS, P.-A. Sept. 1990. "Tramis: concevoir une base de données, c'est la base d'un système d'information". Génie logiciel & Système expert, N° 20, p62-66.

[Casanova & Vidal 83]: CASANOVA, M., AND VIDAL, M. 1983. "Towards a sound view integration methodology". In Proceedings of the 2nd ACM SIGACT/SIGMOD Conference on Principles of Database Systems (Altanta, Ga., Mar. 21-23). ACM New York, pp. 36-47.

[Celis & Deudon 88]: CELIS, M., Deudon, E. 1988. "Intégration de schémas conceptuels ERA", mémoire de fin d'études (2 vol), Facultés universiaires Notre-Dame de la Paix, Namur.

[Chen 76]: CHEN, P.P. mars 1976. "The entity relationship model - Toward a unified view of data", ACM TODS, Vol. 1, N°1.

[Codd 71]: CODD, E.,F. November 1971. "A database sublanguage based on the relational calculus", in proc. ACM-SIGFODET Workshop Data Description, Access and Control, San Diego, CA, pp 35-38.

[Comyn-Wattiau 90]: COMYN-WATTIAU, I. 22 JUIN 1990. "L'intégration de vues dans le système expert SECSI". Thèse de doctorat à l'université Paris VI.

[Dayal & Hwang 84]: DAYAL, U., HWANG, H.Y. Nov 1984. "View definition and generalization for database integration in a multidatabase system". IEEE Transactions On Software Engineering, SE Vol. 10, N°6.

[Elmasri & Wiederhold 79]: ELMASRI, R., WIEDERHOLD, G. 1979. "Data model integration using the structural model". ACM SIGMOD.

[Frantz 90]: FRANTZ, G. July 1990."Microsoft Windows 3.0 Guide du programmeur". Sybex, système d'exploitation.

[Hainaut 86a]: HAINAUT, J.-L. 1986. "Conception assistée des applications informatiques, partie No 2 conception de la base de données". Edition Massons Presses Universitaires de Namur 1986.

[Hainaut 86b]: HAINAUT, J.-L. 20 Août 1986. "Schémas de la base des spécifications". Projet Atelier Bases de Données. Spécification - SPEC-86/8-4.

[Hainaut 88]: HAINAUT, J.-L. 1988. "Introduction à la théorie relationnelle des bases de données". Notes de cours provisoires.

[Hainaut 89]: HAINAUT, J.-L. 1989. "Bases de données et bases de connaissances en gestion des organisations". Notes de cours provisoires.

[Hainaut 90]: HAINAUT, J.-L. 1990. "TRAMIS: a transformation-based database CASE tool". Paper written at the Faculty N-D de la Paix, Namur.

[Jardine 89]: JARDINE, D.A., YAZID, S. 1989. "Integration of Information Submodels". Information Systems Concepts: An In-Depth Analysis, E.D. Falkenberg and P. Lindgreen Eds., North-Holland.

[Khan 79]: KHAN, B. 1979. "A structured logical database design methodology". Ph.D. dissertation, Computer Science Dept., Univ. of Michigan, Ann Arbor, Mich.

[Larson & Al 89]: LARSON, J.A., NAVATHE, S.B., EL-MASRI, R. Avril 1989. "A theory of attribute equivalence in databases with application to schema integration", IEEE Transactions On Software Engineering, Vol. 15, N°4.

[Litwin 84]: LITWIN, W. Apr. 24-27 1984. "MALPHA: A relational multidatabase manipulation language". in proc. IEEE Comput. Soc. First Int. Conf. Data Eng., Los Angeles, CA, pp 86-93.

[Motro 87]: MOTRO, A. Juli 1987. "Superviews: virtual integration of multiple databases", IEEE Transactions On Software Engineering, Vol. 13, N°7.

[Navathe & Gadgil 82]: NAVATHE, S.B., AND GADGIL, S.G. 1982. "A methodology for view integration in logical database design". In Proceedings of the 8th International Conference on Very Large DataBases (Mexico City). VLDB Endowment, Saratoga, Calif.

[Spaccapietra & Parent 90]: SPACCAPIETRA, S., PARENT, C. 1990. "View integration: a step forward", Ecole Polytechnique de Lausanne, laboratoire Bases de données - IN - ECUBLENS.

[Tardieu & Al 79]: TARDIEU, H., NANCI, D., PASCOT, D. 1979. "Conception de système d'information: Conception de la base de données". Editions d'Organisation, Paris.

[Teory & Fry 82]: TEOREY, T., AND FRY, J. 1982. "Design of database structures". Prentice-Hall, Englewood Cliffs, N.J.

[Yao & Al 82]: YAO, S. B., WADDLE, V., AND HOUSEL, B. 1982. "View modeling and integration using the functional data model". IEEE Trans. Softw. Eng. SE-8, 6, 544-553.